



California Enterprise Architecture Program

Service Oriented Architecture

April 21, 2006

Revision History

11/09/2005	Original Draft
11/11/2005	Draft - Replaced California Web Center and CSC with California Service Center and CSC and updated all pictures, updated Introduction, Web Services, ESB, sections as well as California SOA Principle #1.
11/28/2005	Draft - Updated drawings and text in security section.
11/30/2005	Draft - Released to CIO website.
12/06/2005	Draft - Updated RSS diagram.
12/08/2005	Draft – Added Accidental Architecture, Loosely Coupled Interfaces, ETL, Batch Transfers, and FTP, and Information Brokers sections.
1/10/2006	Draft - Added Reference Architecture sections
1/30/2006	Draft - Update Reference Architecture sections
2/27/2006	Draft - Added WSIF definition and Legacy Integration Appendix. Reference Architecture section updated to emphasize legacy integration into services-based platforms. Changed Service Models to Service Patterns. Added Legacy Integration Patterns (Appendix F). Updated Governance & Operation Models.
3/3/2006	Changed Federated Search Engine Pattern to Enterprise Search Pattern. Added “Overview for Business Executives” section. Significant changes to the SOA Management section.
3/10/2006	Added Federated Service Centers Pattern.
3/14/2006	Reviewed and incorporated input from two state working groups, as well as a number of private industry companies.
3/29/2006	Updated Enterprise Search Service and Enterprise Search Engines Patterns.
4/17/2006	Added Foreword section.
4/20/2006	Incorporated Gartner feedback.

Foreword

The California Enterprise Architecture Program (CEAP) would like to recognize the State CIO, Clark Kelso, for his vision and support of this effort.

In the course of developing the Service Oriented Architecture for California, many people have provided direct input, reviewed the proposed SOA architecture, or contributed on a less direct basis. For example, there is a wealth of information on the Web regarding the various components of SOA and the many standards that comprise SOA. CEAP reviewed many documents from standards organizations, industry analysts, the Federal Enterprise Architecture, as well as vendor white papers.

CEAP would like to acknowledge the following companies and organizations that had a significant role in helping to shape the concepts and ideas described in this SOA document:

Private Industry:

IBM, Microsoft, Oracle, SAP, Sonic Software, Autonomy, Cape Clear, Wells Fargo, and Chevron. Gartner, Patricia Seybold Group, W3C, and OASIS.

In addition, CEAP thoroughly appreciated the considerable amount of time (and patience) that various people at IBM, Microsoft, and Gartner provided. Without their help, this would be a less inspiring piece of work.

Public Sector:

Federal Enterprise Architecture, CalPERS, Department of Technology Services, Department of Motor Vehicles, Employment Development Department, Franchise Tax Board, Board of Equalization, Department of Health Services, Department of Transportation, Department of Insurance, Resources Agency, Conservation, Department of Developmental Services, Air Resources Board, and Department of Consumer Affairs.

California Enterprise Architecture Program (CEAP)

Table of Contents

Foreword.....	3
Target Audience	8
Overview for Business Executives.....	9
SOA Introduction	12
<i>The Accidental Architecture</i>	12
SOA	12
<i>Loosely Coupled Interfaces</i>	13
SOA Architecture	15
Web Services.....	15
Service	15
Message.....	16
Dynamic Discovery	16
Web Service Analogy.....	16
Web Service Composition.....	18
Base Services	18
Composite Services	18
Web Service Types	20
SOAP Web services	20
REST Web Services.....	20
Web Services with Presentation Logic	21
Web Service Interfaces.....	21

<i>Web Services Orchestration</i>	<i>22</i>
<i>Web Service Standards</i>	<i>25</i>
<i>Enterprise Service Bus (ESB).....</i>	<i>26</i>
ETL, Batch Transfers, and FTP	27
Information Brokers	27
California SOA Goals.....	30
California SOA Principles.....	31
California SOA Architecture.....	34
<i>California Enterprise Architecture.....</i>	<i>34</i>
<i>Reference Enterprise Architecture.....</i>	<i>34</i>
Enterprise Services.....	36
Shared Services	36
Reference SOA Architecture	38
<i>SOA Service Patterns</i>	<i>39</i>
Application Consuming Web Services Pattern	39
Federated Service Interfaces Pattern	40
Federated Composite Web Services Pattern	41
Federated Service Centers Pattern.....	42
Enterprise Search Service Pattern	43
Federated Search Engines Pattern.....	44
California Service Center (CSC) at DTS Enterprise Service Examples	45
RSS (Real Simple Syndication) Pattern	47

<i>California Business Reference and Service Reference Models</i>	<i>48</i>
<i>California SOA Security Model</i>	<i>49</i>
Introduction	49
XML Security for Web Services	50
Basic Cryptographic Concepts.....	51
Message Integrity and User Authentication with XML Signatures	52
An Introduction to Web Service Security.....	53
Identity and Authentication	55
Identity Authorities Pattern	56
Sharing of Authentication Information.....	56
Security Access Markup Language.....	57
A Citizen Request Example	60
Communities of Interest Pattern.....	61
SOA Firewalls for Web Security.....	62
California SOA Center of Excellence	64
<i>Introduction</i>	<i>64</i>
<i>SOA Excellence Model</i>	<i>65</i>
<i>SOA Management Model.....</i>	<i>66</i>
SOA Centralized Functions	66
SOA Federated Functions	69

<i>Appendix A - Federal SOA</i>	<i>71</i>
<i>Appendix B - Web Service Tenets (Microsoft)</i>	<i>75</i>
<i>Appendix C - SOA Best Practices (IBM)</i>	<i>80</i>
<i>Appendix D - SOA Advantages (Patricia Seybold Group).....</i>	<i>81</i>
<i>Appendix E – WSDL Example</i>	<i>82</i>
<i>Appendix F – Legacy Integration Patterns</i>	<i>83</i>
Overview	83
Integrating Existing Mainframe Apps - Unmodified	83
Placing Web Service Interfaces on Existing Mainframe Apps	83
Compiling COBOL Code into Web Service Languages	83
<i>Appendix G - Definitions</i>	<i>85</i>

Target Audience

This document has a mix of business and technical perspectives. In general, it is targeted at a high-level technical audience. However, it does explore important topics that will be of interest to business leaders.

Business leaders – focused on fielding systems that best support their mission and business needs and achieve the highest return on their IT investments. Note that most of this document is intended for a technical audience. However, please read the “Overview for Business Executives” section.

Chief Architects – who are responsible for the definition and target planning of an Agency or Department’s Enterprise Architecture, working with a variety of architectural implementations.

System/Solution Architects – responsible for building / assembling service components that leverage existing capital assets and business services across the government and industry.

Overview for Business Executives

Big problems require innovative solutions

From the Federal, State and local government levels, there has been an on-going evolution in programs and services that have been established to meet the changing and unique needs of the citizenry. Public perceptions relative to efficiency, effectiveness and the sustained purpose of government coupled with funding realities, has necessitated that all levels of government seek ways to “reinvent” themselves. Public pressure for responsive, accessible, flexible, and accountable services, is forcing government to think more holistically and strategically. The new service paradigm of doing more with less, has necessitated a requirement for government, including its respective agencies, departments and programs, to communicate, collaborate and coordinate for the purpose of eliminating redundant and duplicative programs and for sharing information and services. No where is this more pervasive than in the information technology (IT) arena.

In its infancy IT services in government evolved to solve unique program related business problems, on a project by project basis, with little consideration for a strategic or organizational enterprise architecture approach. Each project solved a unique program related business problem and/or in the process attempted to simplify the government employees work processes. Similar to the evolution of government programs and services, there was little to no coordination and collaboration between internal programs, let alone levels of government, relative to the design, development and deployment of IT related business solutions. As a consequence IT systems sprang up across government as individually developed applications based on unique business service needs that happened to be financed with whatever program funding was available at the time. In the absence of a holistic IT enterprise architecture planning approach to procurement and systems development and acquisition - an approach that would have required governmental IT strategic planning/enterprise architecture guidelines, governance structures with clearly defined principles, industry best practices, policies and procedures - IT systems and IT solutions were built and maintained to support a specific governmental function.

Tightly Coupled Systems mean less flexibility

In the past when business rules changed on legacy systems, either because of program policy changes or new legislative mandates, many time consuming and costly application changes were necessary to implement these changes. In addition, these changes also impacted data bases and a spectrum of system interfaces which also proved to be problematic for both the IT application support staff and business users. As these systems continue to age, we also lose the necessary intellectual resources and vendor support required to adequately maintain and support them. This coupled with the citizen demand for increased responsiveness by providing more access to government information and on-line self-help service options is forcing government to look at more flexible service options that allow for the integration of legacy systems with newer web-based and loosely coupled technologies. Government at all levels recognize that the cost of replacing legacy systems with newer web service applications is not a reality and that to be effective we must be able to leverage existing systems using new open standard integrations approaches. The goal is to reuse and leverage what we have, rather than throw the baby out with the bath water.

What we have today are many monolithic systems that have similar functionality, (such as enrollment functions in health and social services, or revenue systems that reside at Franchise Tax Board, Board of Equalization and Employment Development Department) but with limited interoperability. The interoperability that may be required by law (such as data exchange across organizations) requires that large complex integration systems be built on top of these IT systems. To compound the problem, many of these systems were developed at a time when IT was considered “tightly coupled”, meaning that user interface code, database, business logic, and hardware were developed together and are not easily separated. Basically a government service or program was “tightly coupled” to an IT system that was “tightly coupled” to the various technologies that support them - and in many cases the developers (staff) who support the systems. A COBOL programmer that may have developed an application at FTB or Health Services, or any number of other programs in the state often become mission critical to the actual governmental service that the state performs. These tightly coupled systems were justifiable at the time since computing power was relatively low and costly.

Over the past 10-15 years information technology and how it is used to support “services” has begun to evolve - slowly. The concept of client server and object oriented technologies, in addition to significant advances in computing power with lower costs, have sought to ease the burden of “tightly coupled” systems by allowing the various components of a holistic IT system to be run on separate or “loosely coupled” systems. This allowed the database to be separate from the business logic and the hardware that support it. A database could now be run on many types of hardware, and various database systems to be used in systems that may be developed in various programming languages. It allowed for much greater flexibility and efficiency in how systems were developed and integrated and ultimately in how services to citizens are provided. However, client server systems are still tightly coupled business logic with the user interface.

Citizens Expect More

Citizens have now come to expect that technology will allow them to work across organizational boundaries to gain access to the services they need. Just as easily as people expect to be able to make travel arrangements over the internet that include airfare, hotel and car, people have come to expect an integrated government that will allow them to pay their vehicle registration, pay their taxes, and gain access to services they may need such as unemployment benefits or emergency notifications that may affect their health and welfare, through integrated systems. This requires a loose coupling of business processes. That is, parts of business processes are common and they are candidates for shared services which will reduce code redundancy, lower infrastructure and staff costs, as well as better enforce enterprise standards.

SOA enables loosely coupled services

The expectation of an integrated government and self service “services” necessitate a technical infrastructure that will support these expectations. In the past few years this technology has begun to mature and evolve. The concept of “Service Oriented Architecture” (SOA), and Web Services allows for an orchestration of various business related, technical components that can integrate across lines of government and governmental services. For example, rather than building a payment service for each IT application, the service can be built once and consumed across government. Multiple services can now be combined to form an integrated experience. If for example revenue agencies have a common client, the client will have the ability to authenticate once, and have a look and feel of working from a common Customer Service Center (such as www.taxes.ca.gov) yet satisfying the business requirements of the various departments who provide the web services that underlay the application.

In another example, a citizen who may need temporary assistance will find that they need to interact with many governmental agencies. If for example a mother needs access to food stamps, child care, job information, and possibly educational opportunities, she may need to interact or enroll in programs provided by the county, EDD, community colleges, etc. With an integrated IT system that utilizes SOA, the barriers to organizations begin to disappear. An enrollment for services can now enable enrollment in various programs at the same time, including notifying her of child care opportunities, job notifications, and educational opportunities with the hope of allowing her to be self sufficient in a more rapid manner.

SOA and Web Services are an integral part of helping to achieve the goals of the State IT Strategic Plan, and for providing better integration of business services across government. The adoption of the SOA Architecture, developed by the California Enterprise Architecture Program, is a necessary first step in beginning to leverage technology to solve some very complex business issues. It also lays the foundation for the technical components, infrastructure and standards that need to be implemented for success of this critical technology shift.

SOA Introduction

The Accidental Architecture

Over the past two decades, numerous distributed computing models have arrived on the scene, including DCE, CORBA, DCOM, MM, EAI brokers, J2EE, .NET, and web services. However, indications are that only a small percentage of enterprise applications are connected, regardless of the technology being used. According to a research report from Gartner Inc. ("Integration Brokers, Application Servers and APSs" 10/2002), that number is less than 10%.

Another statistic is even more surprising - of the applications that are connected, only 15% are using formal integration middleware. The rest are using the ETL and batch file transfer techniques, which are largely based on hand-coded scripting and other custom solutions.

The Gartner 15% statistics provides a sobering data point that illustrates the true state of integration today. How are the other 85% of applications connected? A very common situation that exists in enterprises today is what I refer to as "the accidental architecture."

The accidental architecture is something that nobody sets out to create; instead, it's the result of years of accumulating one-of-a-kind pointed integration solutions. In an accidental architecture, corporate applications are perpetually locked into an inflexible integration infrastructure. They continue to be treated as "silos" of information because the integration infrastructure can't adapt to new business requirements.

Most integration attempts start out with a deliberate design, but over time, other pieces are bolted on and "integrated," and the handcrafted integration code drifts away from the original intent. Through incremental patches and bolt-ons, integrated systems can lose their design integrity, especially if the system is maintained by a large number of people to whom the original design intent may not have been well communicated. It's a fact of life that individual point-to-point integrations will drift away from consistency, as engineers make "just this one little change" that's expedient at the time. Eventually, it becomes difficult to even identify the points for making changes, and to understand what the side effects would be as a result. In a deployed system this can lead to disastrous results that will negatively affect your business.

Above excerpts from – *David A. Chappell (Sonic Software) "Enterprise Service Bus" 2004*

SOA

SOA has become a well-known and somewhat elusive acronym. If one asks two people to define SOA one is likely to receive two very different, possibly conflicting, answers. Some describe SOA as an IT infrastructure for business enablement while others look to SOA for increasing the efficiency of IT.

Gartner defines SOA as "an architectural style in which certain discrete functions are packaged into modular, shareable, distributable elements ("services"), which can be invoked by consumers in a loosely coupled manner". With SOA, integration becomes forethought rather than afterthought—the end solution is likely to be composed of services developed in different programming languages, hosted on disparate platforms with a variety of security models and business processes. While this concept sounds incredibly complex it is not new—some may argue that SOA evolved out of the experiences associated with designing and developing distributed systems based on previously available technologies. Many of the concepts associated with SOA, such as services, discovery, and late binding were associated with CORBA and DCOM. Similarly, many service design principles have much in common with earlier OOA/OOD techniques based upon encapsulation, abstraction, and clearly defined interfaces.

The acronym SOA prompts an obvious question—what, exactly, is a service? Simply put, a service is a program that can be interacted with through well-defined message exchanges. Services must be designed for both availability and stability. Services are built to last while service configurations and aggregations are built for change. Agility is often promoted as one of the biggest benefits of SOA—an organization with business processes implemented on a loosely-coupled infrastructure is much more open to change than an organization constrained by underlying monolithic applications that require weeks to implement the smallest change. Loosely-coupled processes and loosely-coupled information structures result in loosely-coupled systems

Services and their associated interfaces are designed to be re-configured or re-aggregated to meet the ever-changing needs of business. Services remain stable by relying upon standards-based interfaces and well-defined messages—in other words, SOAP and XML schemas for message definition. Services designed to perform simple, granular functions with limited knowledge of how messages are passed to or retrieved from it are much more likely to be reused within a larger SOA infrastructure.

SOA and Web Services have recently been used interchangeably. That is because most of the SOA standards work has focused on web services. New standards are emerging and new vendor products are now available that focus on Enterprise Service Bus concepts. For example, major technology companies are currently working on Service Data Objects. SDOs will enable uniform access to application data and a common programming model for all data sources, wherever and however the data is stored. SDOs leverage the simplicity of XML without introducing the complexity of XML Schema or the performance issues of serialization. Using SDOs and SOA together, systems programming tasks are separated from the business logic and encapsulated in reusable services. They simplify business application programming without getting pulled into technology or implementation details.

Loosely Coupled Interfaces

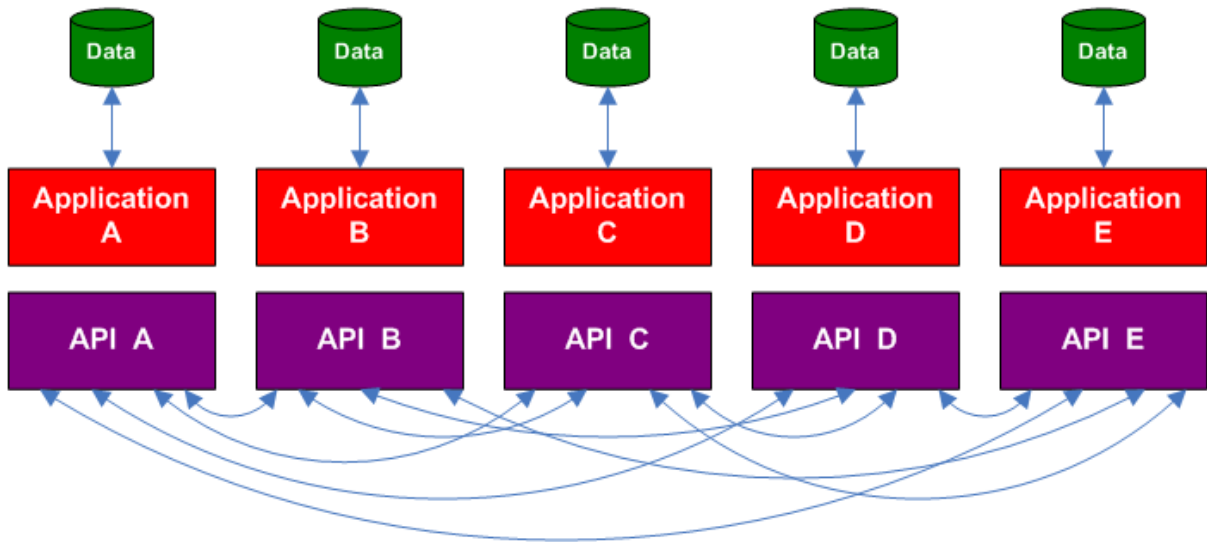
Most current applications interact via tightly coupled interfaces. This requires the calling application to know language specific and datatype details of the target application (for example, Java API). This makes maintenance more difficult and the notion of shared services built on tightly coupled interfaces very difficult.

Loosely coupled interfaces use industry standard XML messages to communicate. This process uses a messaging broker (or backbone) to handle the delivery details. This is often referred to as an Enterprise Service Bus.

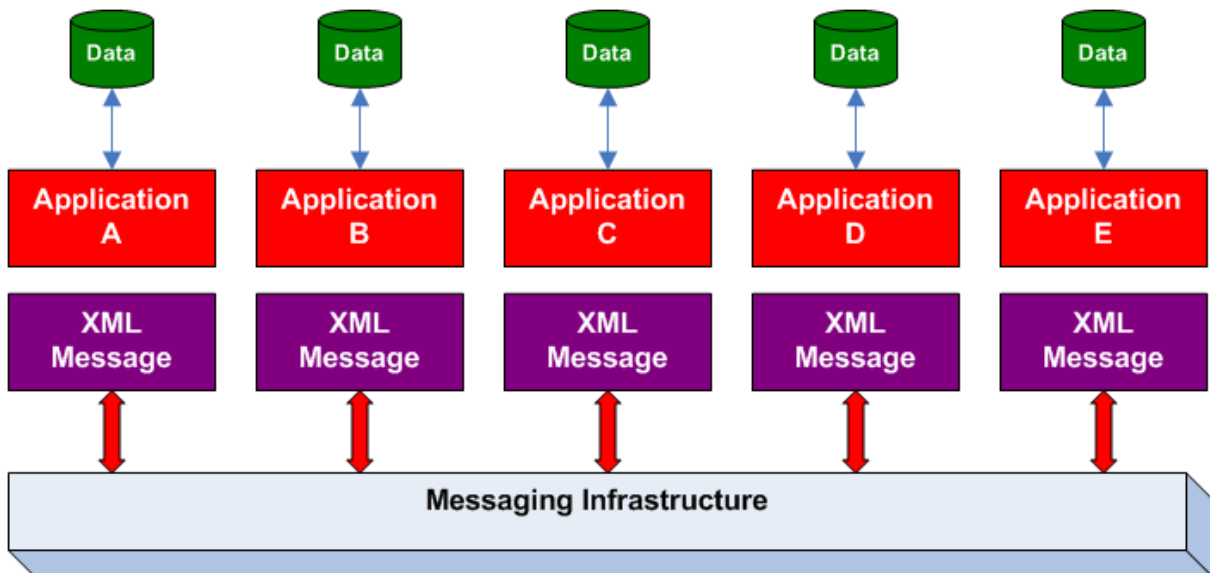
SOA web services are based on loosely coupled interfaces. XML messaging is the core of web services. There are many WS* standards that define the different types of XML content.

The above paragraphs address loose coupling from a technical perspective. It should be noted that business processes and information can (and usually are) designed for only a limited set of consuming applications. Thus, they are tightly-coupled limiting their usefulness.

Tightly Coupled Interfaces



Loosely Coupled Interfaces



SOA Architecture

SOA presents the big picture of what you can do with Web services. Web services specifications define the details needed to implement services and interact with them. However, SOA is an approach to build distributed systems that deliver application functionality as services to end-user applications or to build other services. SOA can be based on Web services, but it may use other technologies instead. In using SOA to design distributed applications, you can expand the use of Web services from simple client-server models to systems of arbitrary complexity.

Thus, individual software assets become the building blocks to develop other applications. You can reduce the complexity of systems by using a common style of interaction that works with both new and legacy code. There is a standard way of representing and interacting with these software assets; now the focus shifts to application assembly based on these building blocks.

Web Services

In order for SOA to be widely adopted, a practical standardized implementation mechanism must exist. Most Web Services are defined in WSDL (XML) and use standard protocols to communicate (SOAP/HTTP). So, using Web Services appears to be the practical solution to implementing an SOA. Alternatively, one could use HTTP-Get or HTTP-Post without SOAP however the data is limited to named-value pairs. SOAP is much more flexible and can handle complex types such as datasets, classes, and other objects.

Web Services are comprised of many components (see [Web Service Standards](#) in this document). Here are a few of the most common. Each of these plays an essential role in SOA.

Service

A service in SOA is an application function packaged as a reusable component for use in a business process. It either provides information or facilitates a change to business data from one valid and consistent state to another. The process used to implement a particular service does not matter, as long as it responds to your commands and offers the quality of service you require.

Through defined communication protocols, services can be invoked that stress interoperability and location transparency. A service has the appearance of a software component, in that it looks like a self-contained function from the service requester's perspective. However, the service implementation may actually involve many steps executed on different computers within one enterprise or on computers owned by a number of business partners. A service might or might not be a component in the sense of encapsulated software. Like a class object, the requester application is capable of treating the service as one.

Web services are based on invocation using SOAP messages which are described using WSDL over a standard protocol such as HTTP. Use of Web services is a best practice when communicating with external business partners.

For example, one might query a web services repository to find a list of services that provide doctor or real estate licensing. In this case, it might return Professional License Service, Medical Doctor License Service, Real Estate License Service, Medical Doctor License Verification Service, Medical Doctor Education Verification Service, etc.

“Individual” services, such as Medical Doctor License Verification Service and Medical Doctor Education Verification Service are built with a granular set of functionality. They can be combined into “composite”

services such as Medical Doctor License Service which is coarse-grained. Or, they can be wrapped to handle requirements that are not included in the service interfaces. (see [Web Service Types](#))

Message

Service providers and consumers communicate via messages. Services expose an interface which defines the behavior of the service and the messages they accept and return. According to Gartner, “a service interface should specify three facets: Identifiers, Formats, and Protocols. This is a network concept that is used to ensure the loose coupling of network components across the global Internet. Identifiers are the names or “addresses” of resources, eg URLs. Formats are the message structures, or in the case of XML, the document structures. And protocols are the rules of interaction between the consumer and provider, eg the message exchange pattern.” Because the interface is platform and language independent, the technology used to define messages must also be agnostic to any specific platform/language. Therefore, messages are constructed using XML documents that conform to XML schema. XML provides all of the functionality, granularity, and scalability required by messages. That is, for consumers and providers to effectively communicate, they need a non-restrictive type of system to clearly define messages; XML provides this.

Because consumers and providers communicate via messages, the structure and design of messages should not be taken lightly. Messages need to be implemented using a technology that supports the scalability requirements of services. While XML interfaces are designed to be extensible, having to redesign entire interfaces due to unanticipated changes will break existing consumers and providers which can prove to be costly.

Dynamic Discovery

Dynamic discovery is an important piece of SOA. At a high level, one searches the registry, gets a URL, and downloads the WSDL file. The directory service is an intermediary between providers and consumers. Providers register with the directory service and consumers query the directory service to find service providers. Most directory services organize services based on criteria and categorize them. Consumers can then use the directory services' search capabilities to find providers. Embedding a directory service within SOA accomplishes the following:

1. Scalability of services; you can add services incrementally.
2. Decouples consumers from providers.
3. Allows for hot updates of services.
4. Provides a look-up service for consumers.
5. Allows consumers to choose between providers at runtime rather than hard-coding a single provider.

Web Service Analogy

Although the concepts behind SOA were established long before web services came along, web services play a major role in a SOA. This is because web services are built on top of well-known, platform-independent protocols. These protocols include HTTP, XML, UDDI, WSDL, and SOAP. It is the combination of these protocols that make web services so attractive. Moreover, it is these protocols that fulfill the key requirements of a SOA. That is, a SOA requires that a service be dynamically discovered and invoked. This requirement is fulfilled by UDDI, WSDL, and SOAP. SOA requires that a service have a platform-independent interface. This requirement is fulfilled by XML. SOA stresses interoperability. This requirement is fulfilled by HTTP. This is why web services lie at the heart of SOA.

Acronyms	Practical Examples
UDDI	Phone Book
WSDL	Contract

SOAP	Envelope
HTTP, SMTP, FTP	Mail person
Programing (Java, Servlet, ASP.NET, C#)	Speech
Schema	Vocabulary
XML	Alphabet

Simplifying Web Service Terms

The basic steps for locating and calling a web service are illustrated in the below drawing (from Patricia Seybold Group).

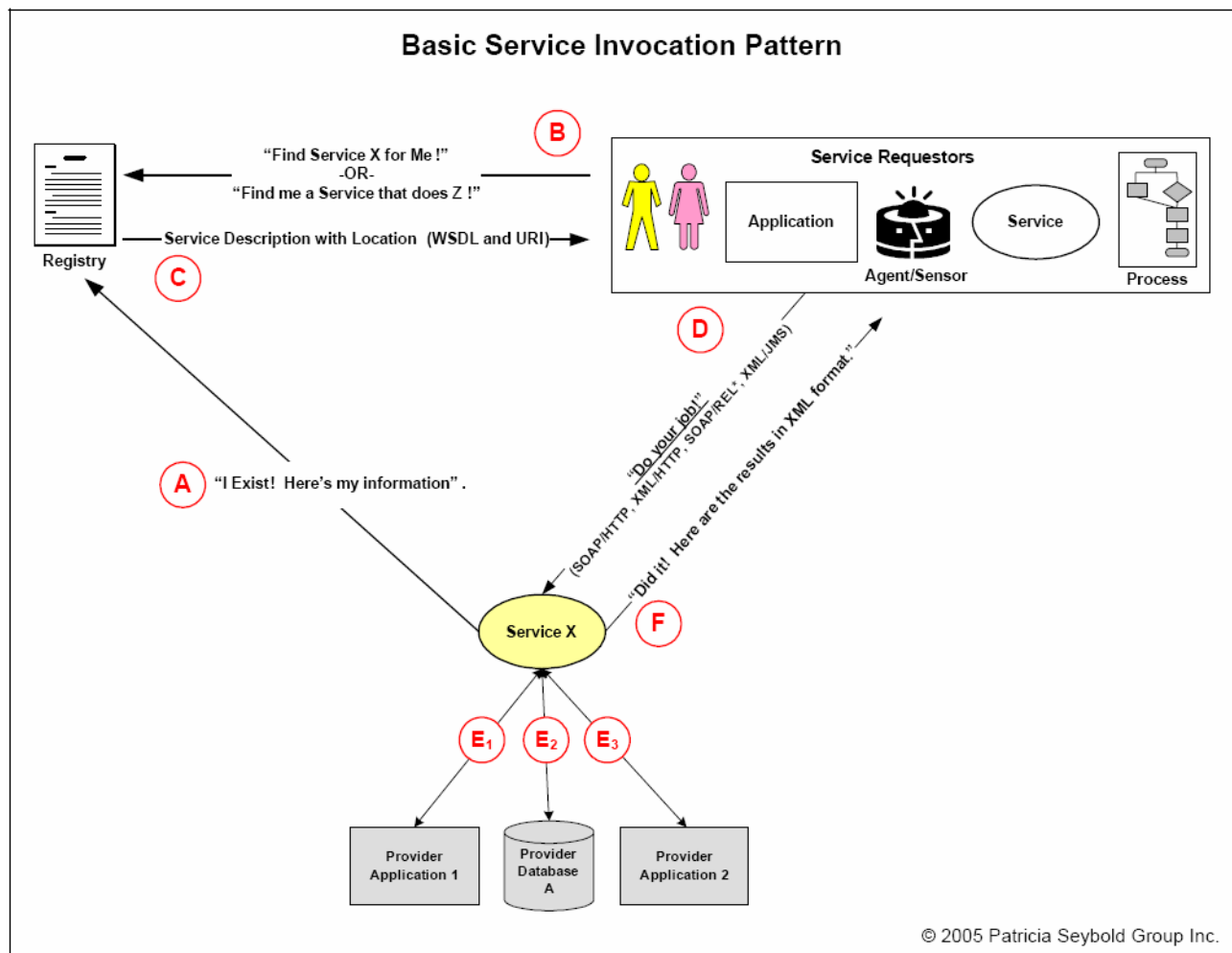
A – Providers register their web services with a common registry, based on a standard such as UDDI. This includes the location and a detailed description of the service in the form of a WSDL XML document.

B – An application (or a different web service) invokes the service from provider A.

C – A SOAP message is sent to the end point of the provider service.

E – The service process the request based on its internal functionality, which is hidden from the external user.

D – The service returns the results in XML format to the requesting application.



Web Service Composition

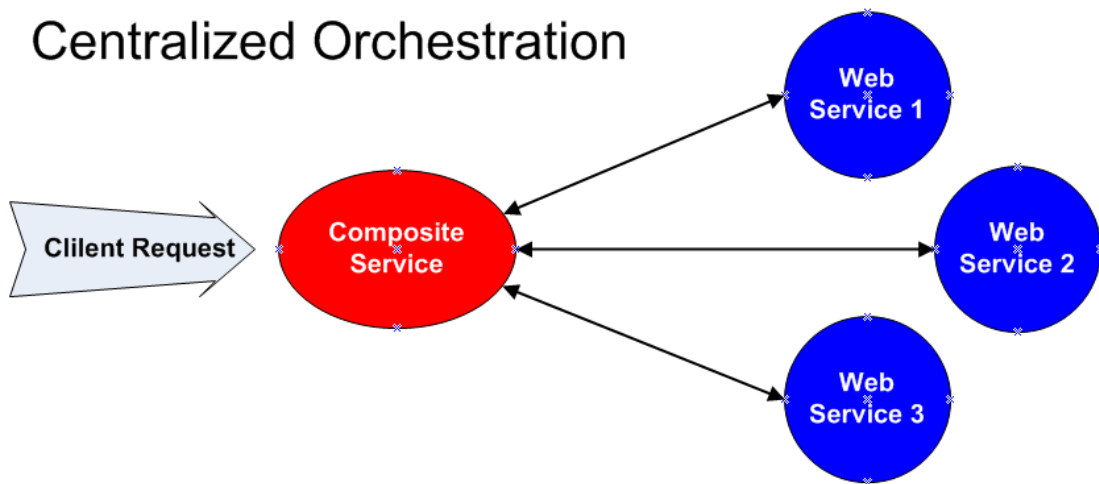
Base Services

Web services encapsulate information, software or other resources, and make them available over the network via standard interfaces and protocols. Web service architectures are based on the notion of building a library of specific *base* services. Generally, the more the more simple and generic the functionality the better the chances are that the service can be used in multiple applications. For example, Address Verification Service, Credit Card Payment Service, and Education Verification Service are candidates for base web services.

Composite Services

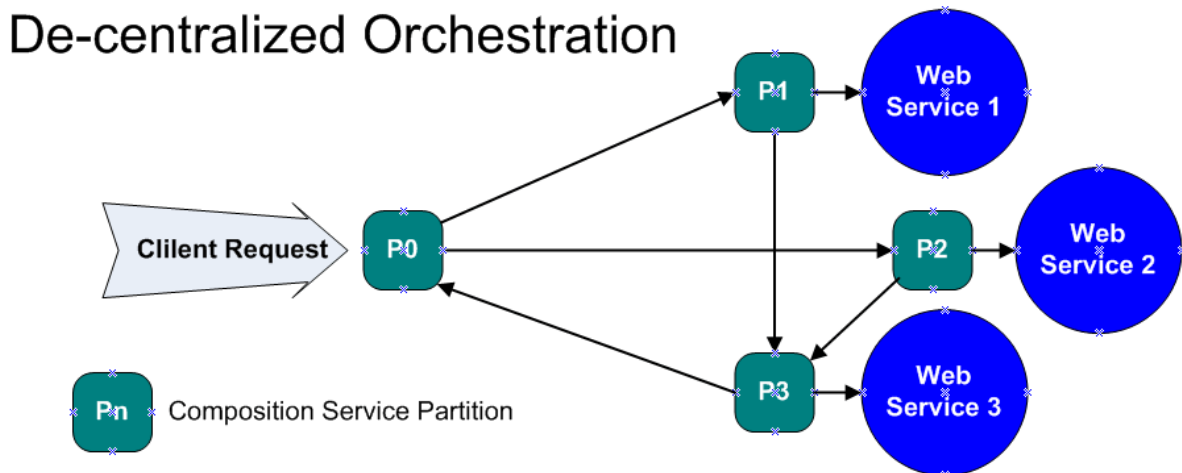
A second powerful notion focuses on aggregating base web services into larger, less granular services. Complex web services may be created by aggregating the functionality provided by simpler ones. This is referred to as service composition and the aggregated web service becomes a *composite* web service. For example, Dentist License Verification Service and Dentist Education Verification Service might be rolled into Dentist Qualifications Service.

Composite web services may be developed using a specification language such as BPEL and executed by a workflow engine. Typically, a composite web service specification is executed by a single coordinator node. It receives the client requests, makes the required data transformations and invokes the component web services as per the specification. This mode of execution is known as *centralized orchestration*.



In *decentralized orchestration* of composite web services, there are multiple engines, each executing a composite web service specification (a portion of the original composite web service specification but complete in itself) at distributed locations. The engines communicate directly with each other (rather than through a central coordinator) to transfer data and control when necessary in an asynchronous manner.

Decentralized orchestration is often referred to as choreography.



Web Service Types

(This section is an excerpt from: XML.COM <http://www.xml.com/pub/a/ws/2003/09/30/soa.html>)

There are two main styles of Web services: [SOAP](#) web services and [REST](#) web services.

SOAP Web services

A SOAP web service introduces the following constraints:

1. Except for binary data attachment, messages must be carried by SOAP.
2. The description of a service must be in [WSDL](#).

A SOAP web service is the most common and marketed form of web service in the industry. Some people simply collapse "web service" into SOAP and WSDL services. SOAP provides "a message construct that can be exchanged over a variety of underlying protocols" according to the [SOAP 1.2 Primer](#). In other words, SOAP acts like an envelope that carries its contents. One advantage of SOAP is that it allows rich message exchange patterns ranging from traditional request-and-response to broadcasting and sophisticated message correlations. There are two flavors of SOAP web services, SOAP RPC and [document-centric SOAP web service](#). SOAP RPC web services are not SOA; document-centric SOAP web services are SOA.

A SOAP RPC web service breaks the second constraint required by an SOA. A SOAP RPC Web service encodes RPC (remote procedure calls) in SOAP messages. In other words, SOAP RPC "tunnels" new application-specific RPC interfaces through an underlying generic interface. Effectively, it prescribes both system behaviors and application semantics. Because system behaviors are very difficult to prescribe in a distributed environment, applications created with SOAP RPC are not interoperable by nature. Many real life implementations have confirmed this.

Faced with this difficulty, both [WS-I basic profile](#) and SOAP 1.2 have made the support of RPC optional. RPC also tends to be instructive rather than descriptive, which is against the spirit of SOA. Ironically, SOAP was [originally designed just for RPC](#). It won't be long before someone claims that "SOAP" actually stands for "SOA Protocol".

REST Web Services

The term [REST](#) was first introduced by Roy Fielding to describe the [web architecture](#). A [REST web service](#) is an SOA based on the concept of "resource". A resource is anything that has a URI. A resource may have zero or more representations. Usually, people say that a resource does not exist if no representation is available for that resource. A REST web service requires the following additional constraints:

1. Interfaces are limited to HTTP. The following semantics are defined:
 - HTTP GET is used for obtaining a [representation](#) of a resource. A consumer uses it to [retrieve a representation](#) from a URI. Services provided through this interface must not incur any obligation from consumers.
 - HTTP DELETE is used for removing representations of a resource.
 - HTTP POST is used for updating or creating the representations of a resource.
 - HTTP PUT is used for creating representations of a resource.
2. Most messages are in XML, confined by a schema written in a schema language such as [XML Schema](#) from W3C or [RELAX NG](#).
3. Simple messages can be encoded with URL encoding.
4. Service and service providers must be resources while a consumer can be a resource.

REST web services require little infrastructure support apart from standard HTTP and XML processing technologies, which are now well supported by most programming languages and platforms. REST web

services are simple and effective because HTTP is the most widely available interface, and it is good enough for most applications. In many cases, the simplicity of HTTP simply outweighs the complexity of introducing an additional transport layer. – *End of XML.COM referenced material.*

One example of a RESTful Web Service is the Yahoo! Local Search service (with results) shown below. It shows the popular practice of encoding all of the information required to invoke the service in the URL. As you can see, the entire interaction took place within the browser. Entering the service's URL in the browser address line returned an XML document with the name and location of pizza shops near our office. The simplicity offered by REST makes it attractive in situations where only simple name-value pairs are required.

RESTful Service: Yahoo! Local Search

The screenshot shows a browser window with the address bar containing the URL: `http://api.local.yahoo.com/LocalSearchService/V1/localSearch?appid=YahooDemo&query=pizza&zip=02109&results=2`. The page content displays an XML document with the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<ResultSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="urn:yahoo:ld" xsi:schemaLocation="urn:yahoo:ld
http://api.search.yahoo.com/LocalSearchService/V1/LocalSearchResponse.xsd" totalResultsAvailable="475" totalResultsReturned="2" firstResultPosition="1">
  <ResultSetMapUrl>http://local.yahoo.com/mapview?stx=pizza&csz=Boston%
  2C+MA&city=Boston&state=MA&radius=5&ed=02Q6D6131Dxp4XaKgLdcqeSr0.x522vVxBGfm1o-</ResultSetMapUrl>
  <Result>
    <Title>Ernesto's Pizza</Title>
    <Address>69 Salem St</Address>
    <City>Boston</City>
    <State>MA</State>
    <Phone>(617) 523-1373</Phone>
    <Rating>/>
    <Distance>0.14</Distance>
    <Url>http://local.yahoo.com/details?
    id=10162605&state=MA&stx=pizza&csz=Boston+MA&ed=yi2Nqa160SziKSDjH7WcUW.3vOi8k8lGNrS7wxE_aRXgbh3XQ06TVDliNZ5hg--</Url>
    <ClickUrl>http://local.yahoo.com/details?
    id=10162605&state=MA&stx=pizza&csz=Boston+MA&ed=yi2Nqa160SziKSDjH7WcUW.3vOi8k8lGNrS7wxE_aRXgbh3XQ06TVDliNZ5hg--</ClickUrl>
    <MapUrl>http://maps.yahoo.com/maps_result?name=Ernesto%
    27s+Pizza&desc=6175231373&csz=Boston+MA&qty=9&cs=9&ed=yi2Nqa160SziKSDjH7WcUW.3vOi8k8lGNrS7wxE_aRXgbh3XQ06TVDliNZ5hg--</MapUrl>
  </Result>
  <Result>
    <Title>Deluna Pizza Delivery</Title>
    <Address>/>
    <City>Boston</City>
    <State>MA</State>
    <Phone>(617) 951-1300</Phone>
    <Rating>/>
    <Distance>0.38</Distance>
    <Url>http://local.yahoo.com/details?id=28474382&state=MA&stx=pizza&csz=Boston+MA&ed=Jpq5yq131DzZcYyPF68JhNK_Is.W00IITW12hAU-</Url>
    <ClickUrl>http://local.yahoo.com/details?id=28474382&state=MA&stx=pizza&csz=Boston+MA&ed=Jpq5yq131DzZcYyPF68JhNK_Is.W00IITW12hAU-</ClickUrl>
    <MapUrl>/>
  </Result>
</ResultSet>
</ResultSet>
<!-- wsdl.search.re2.yahoo.com uncompressed/chunked Mon May 30 08:42:52 EDT 2005 -->
```

Below the XML response, the URL is broken down into its components:

Host Name	Service Name	Version	Method	Query Arguments with Values
api.local.yahoo.com	LocalSearchService	V1	localSearch	?appid=YahooDemo&query=pizza&zip=02109&results=2

In contrast, SOAP would be used where more complex structures are required such as classes and datasets.

Web Services with Presentation Logic

Most web services are designed to handle business logic and data manipulation. However, there is an emerging standard, Web Services for Remote Portlets (WSRP) that is intended to accommodate presentation logic. So, utilizing this standard one could use a web service to generate web page content.

RSS (Real Time Syndication) is an example of REST-style web services with presentation markup in the results. Some people state the REST style of xml presentation has achieved more success on the Web than WSRP.

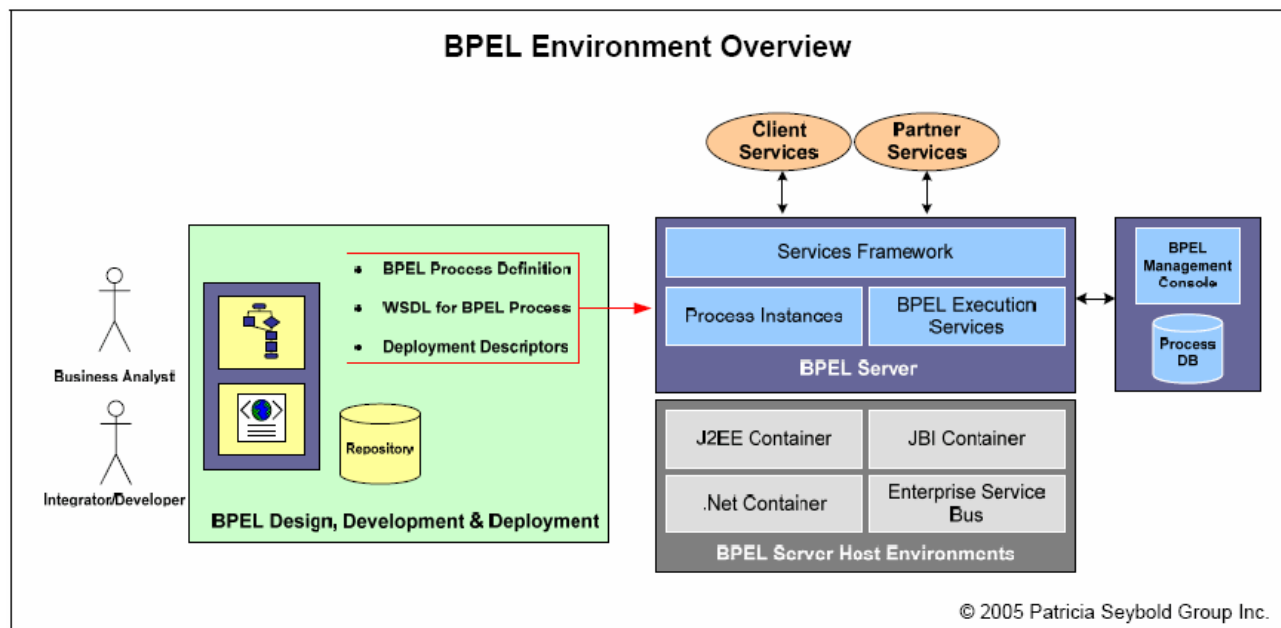
Web Service Interfaces

There are many cases where you might want to specify a web service interface. Multiple web services could then implement the interface ensuring consistency. This design pattern is generally used in federated services. For example, one might create a Professional License interface which would specify the web methods and their details for determining qualifications. This interface could then be implemented by Dentist Licensing, CPA Licensing, and Real Estate Licensing services.

An interface is used when you want to standardize a particular piece of functionality, then apply that functionality to different scenarios.

Web Services Orchestration

An important specification for enterprise integration and service-oriented architecture is business process execution language (BPEL). In BPEL, a business process is a large-grained stateful service, which executes steps to complete a business goal. That goal can be the completion of a business transaction, or fulfilling the job of a service. The steps in the BPEL process execute activities (represented by BPEL language elements) to accomplish work. Those activities are centered on invoking partner services to perform tasks (their job) and return results back to the process. The aggregate work, the collaboration of all the services, is a service orchestration.



It is important to understand the best uses (and limitations) of BPEL. BPEL offers a nice model to abstract orchestration logic from the participating services, and configuration using BPEL over (hard core) coding of service inter-actions is enticing. However, there is processing overhead and infrastructure expense, so BPEL might not be the best choice for simple orchestrations. As a rule of thumb, a simple orchestration is comprised of two to five services and has static interaction patterns.

As a language to develop processes, BPEL is good at executing a series of activities, which occur over time, and interact with internal and external services. These processes may represent IT scenarios, such as integration, or business scenarios, such as information exchange, or flows of work.

As for limitations, BPEL does not account for humans in a process, so BPEL doesn't provide workflow - there are no concepts for roles, tasks and inboxes. In addition, BPEL does not support really complex

business processes, which evolve during their execution, branching out to incorporate new parties and activities. Lastly, BPEL does not have native support for business activity monitoring (BAM). There isn't a data model for measurement and monitoring.

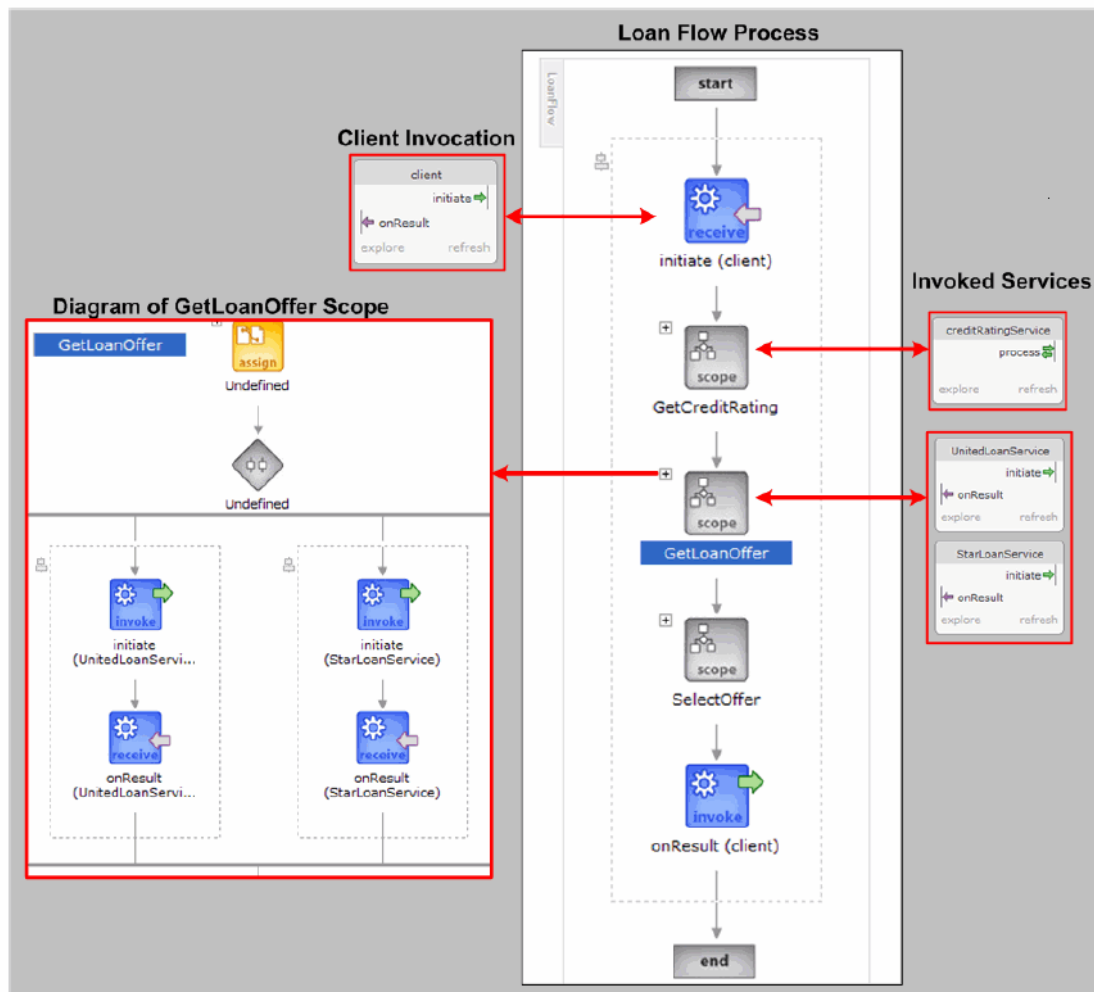
Here is a modified simple Loan Flow from Oracle's BPEL demonstration samples. The business flow is as follows:

1. A customer requests a loan quote
2. The loan flow process starts and performs a customer credit check

The following picture shows the process from a BPEL designer's view. In the center of the diagram is the Loan Flow process. To the right of the loan flow are the invoked (partner) services. At the top left is the client invocation <receive>.

Using a BPEL design tool, you could click on any of the elements to expand the information and make modifications. Shown is an expansion of the GetLoanOffer scope. Here are the invocations of the lending institutions' services. The structure for the invocations is <flow>, indicating concurrent re-quests. When both institutions have replied, the GetLoanOffer scope completes, and processing continues to the SelectOffer scope.

Loan Flow Example, BPEL Designer's View



This diagram shows the Loan Flow example from a BPEL designer's view. The diagram is an annotated composite of screen shot clips from Oracle's BPEL PM tool.

The source code view of this process is shown next. Think of this as the toggle from the design view. The code is a snippet of the actual process, focusing on the partner links and GetLoanOffer scope. In the partner links, you'll notice the first entry is for our BPEL process, with a myRole of "LoanFlowProvider," and partnerRole of requestor. The called services partner links follow; note the roles are switched. The myRole is provider, and partnerRole is requestor.

For each partner link, the partnerLinkType element points to the specific operation to be invoked, as described by the WSDL (portType) of the individual service.

Looking at the GetLoanOffer scope, you can see nested structured activities. There is an outer <sequence> to assign the request data prior to making the invocations, and an inner <flow> that allows the invocations to be concurrent.

The final activity in the process is to provide the requester with the loan offer information.

Loan Flow Example, BPEL Source Code Snippet

```
- <process name="LoanFlow" targetNamespace="http://samples.otn.com" suppressJoinFailure="yes" xmlns:tns="http://samples.otn.com"
  xmlns:services="http://services.otn.com" xmlns:auto="http://www.autoloan.com/ns/autoloan" xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-
  process/" xmlns:bpm="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
  <partnerLinks>
    <!-- first partnerLink is for the LoanFlow Process, it is a service that provides the LoanFlow -->
    <partnerLink name="client" partnerLinkType="tns:LoanFlow" myRole="LoanFlowProvider" partnerRole="LoanFlowRequester" />
    <!-- subsequent partnerLinks are for the services the process invokes -->
    <partnerLink name="creditRatingService" partnerLinkType="services:CreditRatingService" partnerRole="CreditRatingServiceProvider" />
    <partnerLink name="UnitedLoanService" partnerLinkType="services:LoanService" myRole="LoanServiceRequester" partnerRole="LoanServiceProvider" />
    <partnerLink name="StarLoanService" partnerLinkType="services:LoanService" myRole="LoanServiceRequester" partnerRole="LoanServiceProvider" />
  </partnerLinks>
  <variables>
    <!-- abridged variable section, only contains the input of this process -->
    <variable name="input" messageType="tns:LoanFlowRequestMessage" />
    <variable name="crInput" messageType="services:CreditRatingServiceRequestMessage" />
    <variable name="crOutput" messageType="services:CreditRatingServiceResponseMessage" />
    <variable name="crError" messageType="services:CreditRatingServiceFaultMessage" />
  </variables>
  <sequence>
    <!-- sequence of entire process -->
    <!-- client invocation, creates new process instance -->
    <receive name="receiveInput" partnerLink="client" portType="tns:LoanFlow" operation="initiate" variable="input" createInstance="yes" />
    <!-- scope for GetLoanOffer, GetCreditRating Scope and SelectOffer Scope are not shown -->
    <scope name="GetLoanOffer" variableAccessSerializable="no">
      <sequence name="askloanproviders">
        <assign>
          <copy>
            <from variable="input" part="payload" />
            <to variable="loanApplication" part="payload" />
          </copy>
        </assign>
        <flow name="AskLoanProviders">
          <!-- flow allows for concurrent activity processing (both loan service invocations) -->
          <sequence name="askloanproviders" xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
            <invoke name="invokeUnitedLoan" partnerLink="UnitedLoanService" portType="services:LoanService" operation="initiate"
              inputVariable="loanApplication" />
            <!-- ask UnitedLoan for offer, receive reply into loanOffer1 -->
            <receive name="receive_invokeUnitedLoan" partnerLink="UnitedLoanService" portType="services:LoanServiceCallback" operation="onResult"
              variable="loanOffer1" />
          </sequence>
            <sequence name="askloanproviders">
              <!-- ask StarLoan for offer, receive reply into loanOffer2 -->
              <invoke name="invokeStarLoan" partnerLink="StarLoanService" portType="services:LoanService" operation="initiate"
                inputVariable="loanApplication" />
              <receive name="receive_invokeStarLoan" partnerLink="StarLoanService" portType="services:LoanServiceCallback" operation="onResult"
                variable="loanOffer2" />
            </sequence>
          </flow>
        </sequence>
      </scope>
      <!-- end of GetLoanOffer scope -->
      <!-- at end of process, requestor receives reply via a callback -->
      <invoke name="replyOutput" partnerLink="client" portType="tns:LoanFlowCallback" operation="onResult" inputVariable="selectedLoanOffer" />
    </sequence>
    <!-- end of main sequence -->
  </process>
```

PartnerLinks - Services

GetLoanOffer Scope

Web Service Standards

Web services are still evolving and as a result there are a large number of standards. It is likely that some of the standards will be combined. But for now, here is a quick list of some of the more important ones:

Standards Organizations:

W3C - World Wide Web Consortium <http://www.w3.org/>

HTTP, CSS, SOAP, XML, XPath, XSL, WSDL, WS-Addressing, WSCI, WS Choreography Model, plus others.

OASIS – Organization for the Advancement of Structured Information Standards <http://www.oasis-open.org/home/index.php>

WS-BPEL, ADVL, CAP, DSML, ebXML, XACML, SAML, SPML, UDDI, UBL, WS-Reliability, WSRP, WS-Security, WSDM plus others.

WS-I – Web Services Interoperability Organization - Provides interoperability standards in the form of Profiles. <http://www.ws-i.org/> Current profiles include:

- Basic Profile (V1.0, V1.1, Simple SOAP Binding Profile 1.0)
- Attachments Profile 1.0
- Basic Security Profile (V1.0, Security Scenarios)

Standards:

SOAP – Originally: Simple Object Application Protocol. Now, informally referred to a SOA Protocol.

XML – eXtensible Markup Language

WSDL – Web Services Description Language

UDDI – Universal Description Discovery Integration

WSIL – Web Services Inspection Language, may eventually replace UDDI.

WS-Reliability & WS-ReliableMessaging

WSRP – Web Services for Remote Portlets

Process Standards:

BPEL – Business Process Execution Language (Microsoft, IBM). Note, now OASIS standard.

WSCL – Web Services Conversation Language (HP)

WSCI – Web Services Choreography Interface (BEA, Intalio, SAP, Sun)

BPML – Business Process Modeling Language (W3C)

BPSS – Business Process Specification Schema (ebXML)

WSFL – Web Services Flow Language (IBM)

XLANG - (Microsoft)

Transaction Standards:

WS-Transaction (WS-BusinessActivity and WS-AtomicTransaction).

WS-Coordination

Security Standards:

WS-Security

WS-Trust

WS-Provisioning

WS-Federation

WS-Authorization

WS-Policy

WS-Privacy

SAML (Secure Access Markup Language)

STS (Secure Token Service)

Enterprise Service Bus (ESB)

Connecting systems and automating business processes are strong drivers to reducing costs, improving operational efficiency, and capturing new business opportunities. For these reasons, technologies that facilitate integration are a high priority for many technology executives.

While no single product or architecture satisfies all connected system scenarios, there are a variety of established options in the market today including ETL (Extract Transform Load), EAI (Enterprise Application Integration), and B2B (Business to Business) technologies. More recently, several emerging trends and technologies have expanded the potential number of integration scenarios covered by a single product offering. These include business process management (BPM), which builds on the existing EAI and

B2B stacks with capabilities such as business activity monitoring, business process orchestration, and rules; and Web services, which provide industry standards for secure, reliable, transacted communication across platforms.

In a tangential play to the expanding functionality of BPM and the proliferation of Web services, a number of traditional EAI and message-oriented middleware (MOM) vendors have begun marketing products under the term "Enterprise Service Bus (ESB)." Introduced in 2002 by Sonic Software and subsequently touted by analysts as a strategic investment, the ESB term has in recent years permeated the IT vernacular of many organizations looking for a new "magic bullet" to the ongoing challenge of connecting systems.

ETL, Batch Transfers, and FTP

Extract, Transform, and Load (ETL) techniques such as FTP file transfers and nightly batch jobs are still the most popular means of integration today.

This often involves nightly dump-and-load operations on data that sits in various applications. The problem is that there is great potential for data to get out of sync between systems. The recovery process from failure of a dump-and-load can sometimes take more than a day to reconcile.

Other issues are associated with nightly batch processing as well. Due to the latency of nightly batch jobs, the best-case scenario is a 24-hour turnaround time when analyzing critical business data. This delay can severely hinder your ability to react to business events in a timely manner.

Sometimes, the end-to-end processing crossing multiple batch-oriented systems can take up to a whole week to complete. The overall latency involved in the processing of data from the source to the target can prevent you from collecting meaningful data that can provide insight into your current business situation.

Information Brokers

Hub-and-spoke integration brokers, or EAI hubs, offer alternatives to the accidental architecture. Integration brokers have been in existence since the middle of the late '90s, and are built upon MOM backbones or application server platforms. Some of the companies in the integration-broker market include:

SeeBeyond, webMethods, Asential (Mercator), Vitria, IBM, TIBCO, BEA.

Integration brokers can help with the "accidental architecture" by providing centralized routing between applications, using a hub-and-spoke architecture. Furthermore, they also the separation of business processes from the underlying integration code through the use of Business Process Management (BPM) software.

However, there are drawbacks to the integration broker approaches. A hub-and-spoke topology doesn't allow regional control over local integration domains. BPM tools that are built on top of a hub-and-spoke topology can't build choreography or business processes that can span departments or business units. The integration broker may be limited by an underlying MOM in its ability to cross physical network LAN segment boundaries and firewalls.

ESBs are based on lessons learned from integration brokers and best practices from standards-based infrastructure based on XML, web services, reliable asynchronous messaging, and distributed components. These collectively form an architecture for a highly distributed, loosely coupled integration fabric to deliver all the key features of an integration broker, but without the barriers.

ESB Definitions

The recent buzz around ESBs is rivaled only by the ambiguity with which the term is defined. While Sonic Software and Gartner originally used the term to refer to the XML-enabled SonicXQ MOM product (which was later renamed "SonicESB"), ESB has also been used to refer to the message bus architectural integration pattern.

Adding to the confusion around the Enterprise Service Bus are the divergent definitions of the product category:

"A Web-services-capable infrastructure that supports intelligently directed communication and mediated relationships among loosely coupled and decoupled biz components." - *Gartner Group*

"The ESB label simply implies that a product is some type of integration middleware product that supports both MOM and Web services protocols." - *Burton Group*

"A standards-based integration backbone, combining messaging, Web services, transformation, and intelligent routing." - *Sonic Software*

"An enterprise platform that implements standardized interfaces for communication, connectivity, transformation, and security." - *Fiorano Software*

"A system architecture in which applications are integrated using service interactions that are loosely-coupled and well-defined to support interoperability, and to enable flexibility and re-use." - *IBM*

"The Enterprise Service Bus is a uniform service integration architecture of infrastructure services that provides consistent support to business services across a defined ecosystem. The ESB is implemented as a service oriented architecture using Web Service interfaces." - *CBDI*

Peter Linkin, senior director of product marketing for BEA's WebLogic puts the problem which ESB's are intended to solve into perspective: "What was needed was a system that just told us what the message was, what the contents were, where it should go, and what the quality of service for it should be. Then, that was handed off to the next level which is something like a central post office. The postmaster says "send me all your messages from all these outpoints, and I will intermediate. I'll make sure they get sent individually and reliably to all the end points." It's a message broker that's driven by business process. The end points don't have to know about each other so there's ignorance at each end and the logic of the business process is in the plumbing."

Common Characteristics

A variety of vendors now consider themselves players in the ESB space, including Sonic Software, Systinet, Tibco, Fiorano, Cape Clear, and IONA. IBM Websphere ESB provides extensive support for connecting a wide range of applications, protocols, and platforms including Web Services messaging infrastructure and mainframe connectivity. BEA Aqualogic Services Infrastructure includes ESB components. Microsoft has bundled ESB functionality into its BizTalk Server. Additionally, Windows Communications Framework will be included in "Windows Vista" which is Microsoft's next generation web services technology. It will provide framework services that support WS-Addressing, MTOM, WS-Policy, WS-Security, WS-Trust, WS-SecureConversation, WS-ReliableMessaging, WS-AtomicTransaction, and WS-Coordination.

While there is no industry-standard definition of the ESB, a common set of characteristics apply to many of the products in this category:

- **Open Standards.** Open standards refers to both the ESB solution components (runtime container, messaging infrastructure, integration services, design-time notations) and the mechanisms for integrated resources to participate (attach, request, respond) on the bus.
- **Message-Based.** The communication mechanism of an ESB is messaging, using standard message notation, protocols, and transports.
- **Distributed.** The ESB runtime environment can be distributed across a networked environment for the purposes of quality of service, quality of protection, and economics.
- **Routing, Invocation, and Mediation.** Routing, invocation, and mediation are the basic functions of the ESB. Routing includes addressability and content based routing. Invocation refers to the ability to make requests and receive responses from integration services and integrated resources. Mediation refers to all translations and transformations between disparate resources including security, protocol, message notation/ format, and message payload (data/semantics).
- **Facilitate.** The ESB must coordinate the interactions of the various resources and provide transactional support.
- **Reliable.** The ESB must guarantee message delivery.

In simple terms, ESBs:

- Route messages between services.
- Convert transport protocols between requestor and service.
- Transform message formats between requestor and service.
- Handle business events from disparate sources.

Some ESB vendors include additional features:

- Service composition
- Business process management

California SOA Goals

1. Provide the blueprint for a service oriented architecture that supports California business services and incorporates IAP concepts.

2. Provide a key set of SOA principles.

[#California SOA Principles](#)

3. Show how SOA fits into the California Enterprise Architecture model.

[#California Enterprise Architecture](#)

[#California E-Gov Business Services Metamodel](#)

4. Establish a California SOA Center of Excellence to provide SOA leadership, governance, and management of SOA components.

[#California Enterprise Architecture Center of Excellence](#)

California SOA Principles

1. **Design for Ease of Use:** Make it easy for your business solution builders to assemble services into applications and business scenarios. Organize the structure of the California Enterprise Repository so it can be easily searched, learned, and managed.
2. **Design web services with appropriate granularity.** The granularity of operations is an important design point. The use of coarse-grained interfaces for external consumption is recommended, whereas fine-grained interfaces might be used inside the enterprise. A coarse-grained interface might be the complete processing for a given service, such as `SubmitPurchaseOrder`, where the message contains all of the business information needed to define a purchase order. A fine-grained interface might have separate operations for: `CreateNewPurchaseOrder`, `SetShippingAddress`, `AddItem`, and so forth.

While the fine-grained interface offers more flexibility to the requester application, it also means that patterns of interaction may vary between different service requesters. This can make support more difficult for the service provider. A coarse-grained interface guarantees that the service requesters will use the service in a consistent manner. SOA does not require the use of coarse-grained interfaces, but recommends their use as a best practice for external integration. Service choreography can be used to create a coarse-grained interface that runs a business process consisting of fine-grained operations.

Granularity can be viewed from several perspectives. One perspective is the amount of data sent/received. A second perspective is complexity of the interface. A third perspective is the number of interactions required to complete a session. An example might be a service that provides all registered voters in a county. Should it send one huge list (one interaction, but a huge dataset); or send just the As then the Bs then the Cs (26 interactions for consuming processes that do indeed need ALL names, but smaller data sets); or one by one (eg by citizen; which could lead to thousands of interactions, with very small data sets). The number of interactions required to complete a "session" can also be driven by the number of steps in a composite operation, which is the example used here.

Gardner recommends designing services to be as general-purpose as possible. Thus if a large number of consumers would use an "all in one" `SubmitPurchaseOrder`, then create it. But provide some "override" services so that consumers with specialized needs can override the generic operation.

3. **Reassemble before Rewrite.** Individual web services can be assembled into composite web services. Standard web interfaces can also be used to quickly create new services. Consider reassembling existing base web services before writing new web services. For example, Federated Jobs Service is a composite of Available Jobs Service and Process Job Application Service.
4. **Web Services should be loosely coupled and extensible.** The binding from the service requester to the service provider should loosely couple the service. This means that the service requester has no knowledge of the technical details of the provider's implementation, such as the programming language, deployment platform, and so forth. The service requester typically invokes operations by way of messages -- a request message and the response -- rather than through the use of APIs or file formats.

“Extensibility is essential to the rapid and efficient evolution of web service interfaces. If every enhancement of a provider interface requires the redeployment of a corresponding consumer interface to the thousands of existing consumers, then change management will grind to a halt. The key is to architect web service interfaces using XML and XSD extensibility mechanisms to enable both forward and backward compatibility between consumer and provider interfaces to loosely couple consumer versions from provider versions”. – *Gartner 2006*

5. **Web Services must have well-defined interfaces.** The service interaction must be well-defined. Web services Description Language (WSDL) is a widely-supported way of describing the details required by a service requester for binding to a service provider. The service descriptions focus on operations used to interact with the following:
 - a. A service
 - b. Messages to invoke operations
 - c. Details of constructing such messages
 - d. Information on where to send messages for processing details of constructing such messages

WSDL should not include any technology details of the implementation of a service. The service requester neither knows nor cares whether the service is written in Java code, C#, COBOL, or some other programming language. It can describe a SOAP invocation using HTTP. Because of its extension mechanisms, it can also define other styles of interaction such as XML content delivered via JMS, direct method calls, calls handled by an adapter that manages legacy code (CICS), and so forth.

The common definition for WSDL allows development tools to create common interfaces for various styles of interaction, while hiding the details of how it invokes the service from the application code. The Web Services Invocation Framework (WSIF), for example, exploits this capability by allowing a run-time determination of the best way to invoke a quality service if the service is exposed in more than one interaction style. See <http://ws.apache.org/wsif/> for WSIF details.

6. **Design stateless base web services.** Services should be independent, self-contained requests, which do not require information or state from one request to another when implemented. Services should not be dependent on the context or state of other services. When dependencies are required, they are best defined in terms of common business processes, functions, and data models, not implementation artifacts (like a session key). Of course, requester applications require persistent state between service invocations, but this should be separate from the base service. Web applications and composite web services can both handle state.
7. **Implement business processes via orchestrating web services into a process flow (BPEL standard).** Some business processes can be implemented in a process flow and called from an application (which implements the entire business process). The individual nodes within the process flow can call other web services, call out to a business rules engine, or call a native API (such as Java or .NET). Process flows also manage state, which means data created by one node is available to other nodes to view, add to, or modify. Additionally, process flow engines (vendor specific) have built in mechanisms to recover a process flow should a system or process failure occur.

For example, a Professional License Application might call several web service process flows (Gather Qualifications, Process Qualifications, Handle Payment, and Create License) to achieve the business process functionality.

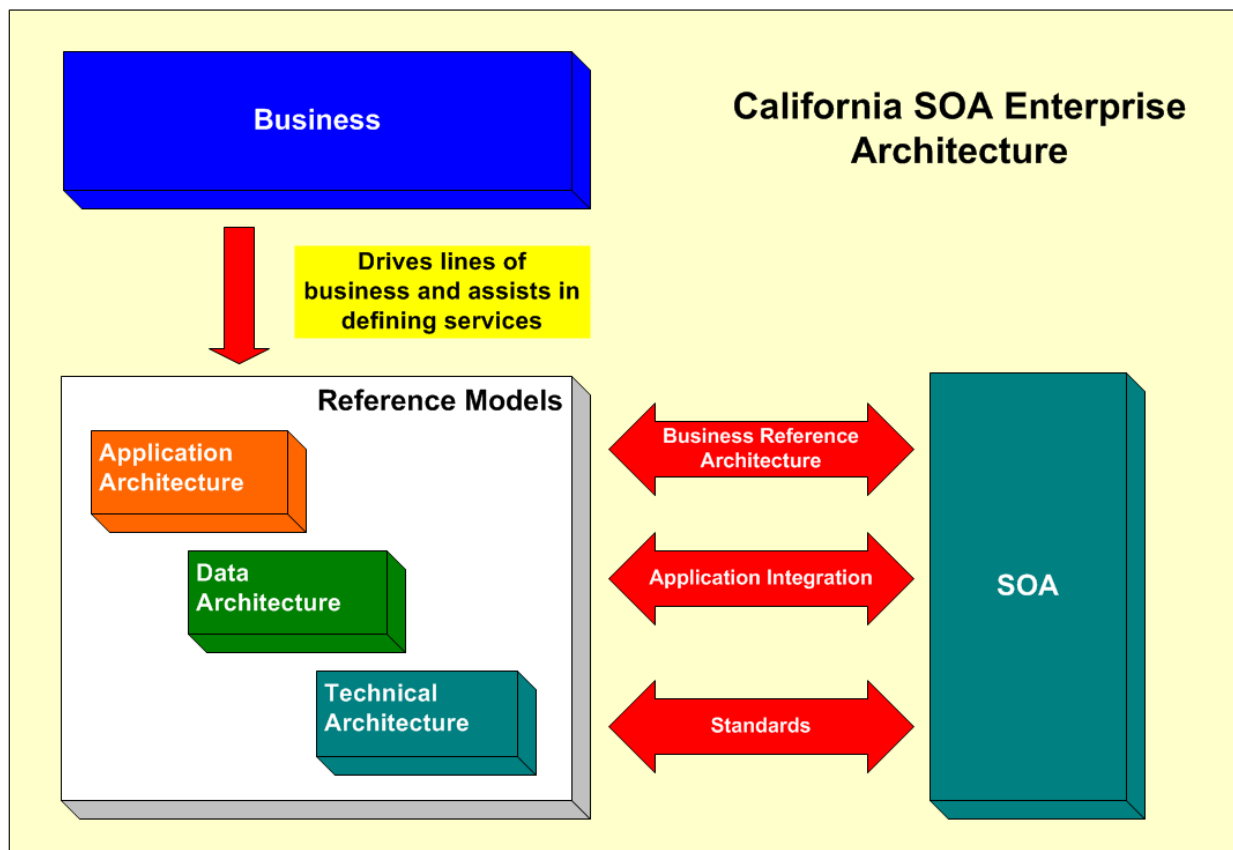
8. **A governance structure must be created to manage web service development and operational environments.** By definition, web services are created with the enterprise in mind. That implies a strong collaboration environment must exist where interested parties agree on how web services will be defined, built, implemented, deployed, supported, enhanced, and managed in production environment. Additional budgets, people, tools, and equipment resources must be allocated appropriately.
9. **Implement web service security and policy enforcement standards:** Liberty Alliance and OASIS have defined a large number of web service security standards. Eventually, the work of both groups will probably be merged into a single set of standards. WS-Security seems to be the most widely used while many other standards within WS* are still evolving. However, this should not deter security mechanisms from being designed into web services.

California SOA Architecture

California Enterprise Architecture

The primary goal of the California Enterprise Architecture Program (CEAP) is to define a blueprint for establishing a customer-centric, flexible, business services driven enterprise architecture that is based on principles, standards, and industry best practices. SOA is a component of the overall Enterprise Architecture since it enables business services to be integrated across organizational boundaries that previously required extensive programming and testing before deployment. It also provides a reference model for application development, and an integration strategy for data and legacy applications.

The SOA architecture is built upon standards based infrastructure. The various components of the SOA architecture help to build out sections of the California Enterprise Architecture Reference Models and, with other technology implementation, help to populate the over all target architecture for California. SOA is only one component of a much larger IT infrastructure that comprises the overall EA.



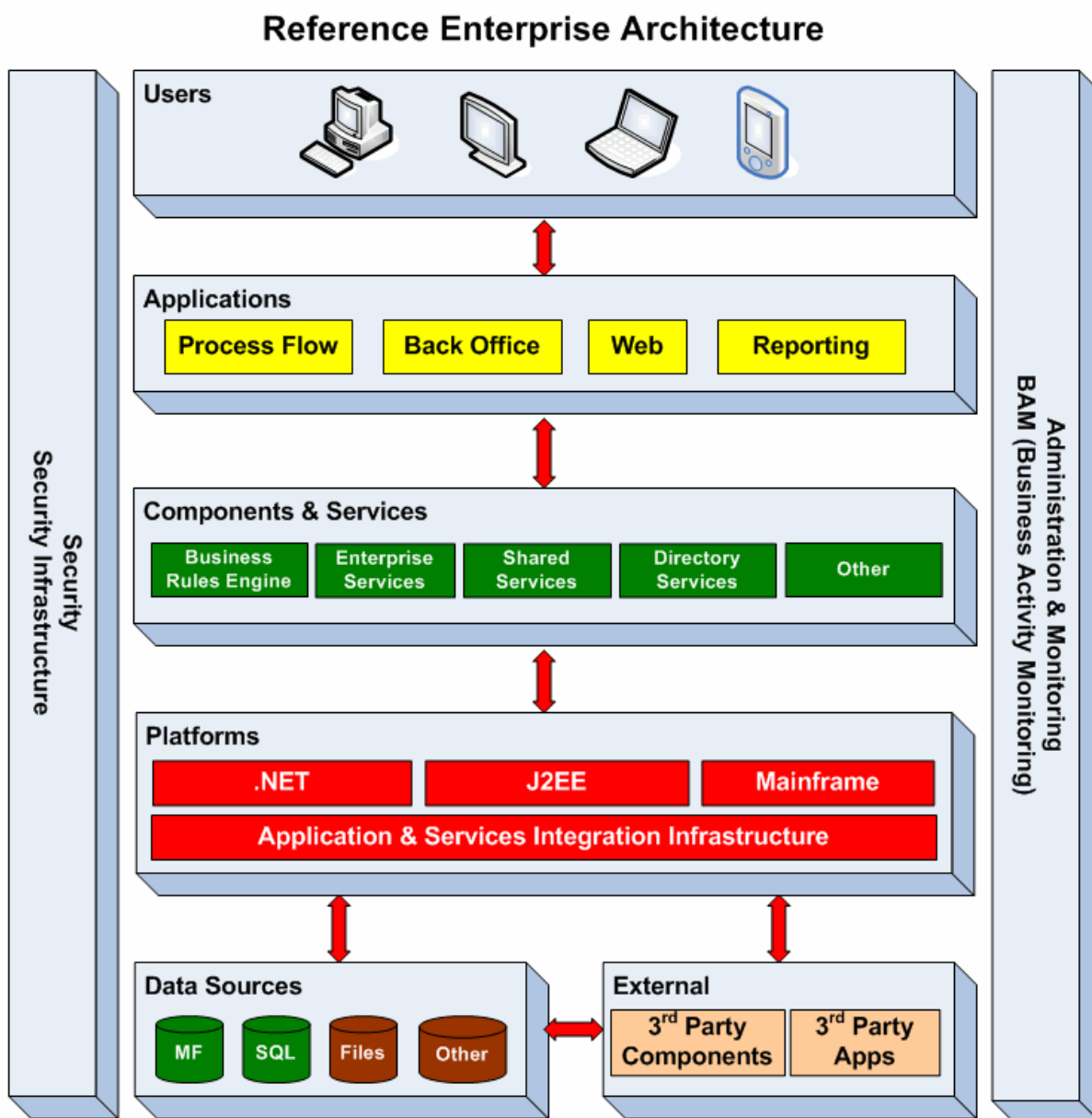
Reference Enterprise Architecture

This section defines the architecture to strive for. It is intended to be a guideline for technical designers, application architects and technical managers. It suggests using components (particularly web service-based)

where possible as well as reducing the number of platforms to three (.NET, J2EE, and Mainframe). This means migrating older applications to either .NET or J2EE platforms while integrating existing mainframe applications into these newer platforms. (See [Appendix F Integrating Legacy Patterns](#)). Within existing constraints, business logic that has common functionality across applications should be moved to .NET and J2EE based applications where it can be implemented as shared services.

A new type of infrastructure will be required to support a services-based environment. This means new messaging, routing, XML transforming, and security components. A new business activity monitor (BAM) tool would be quite useful in keeping track of service component performance and availability.

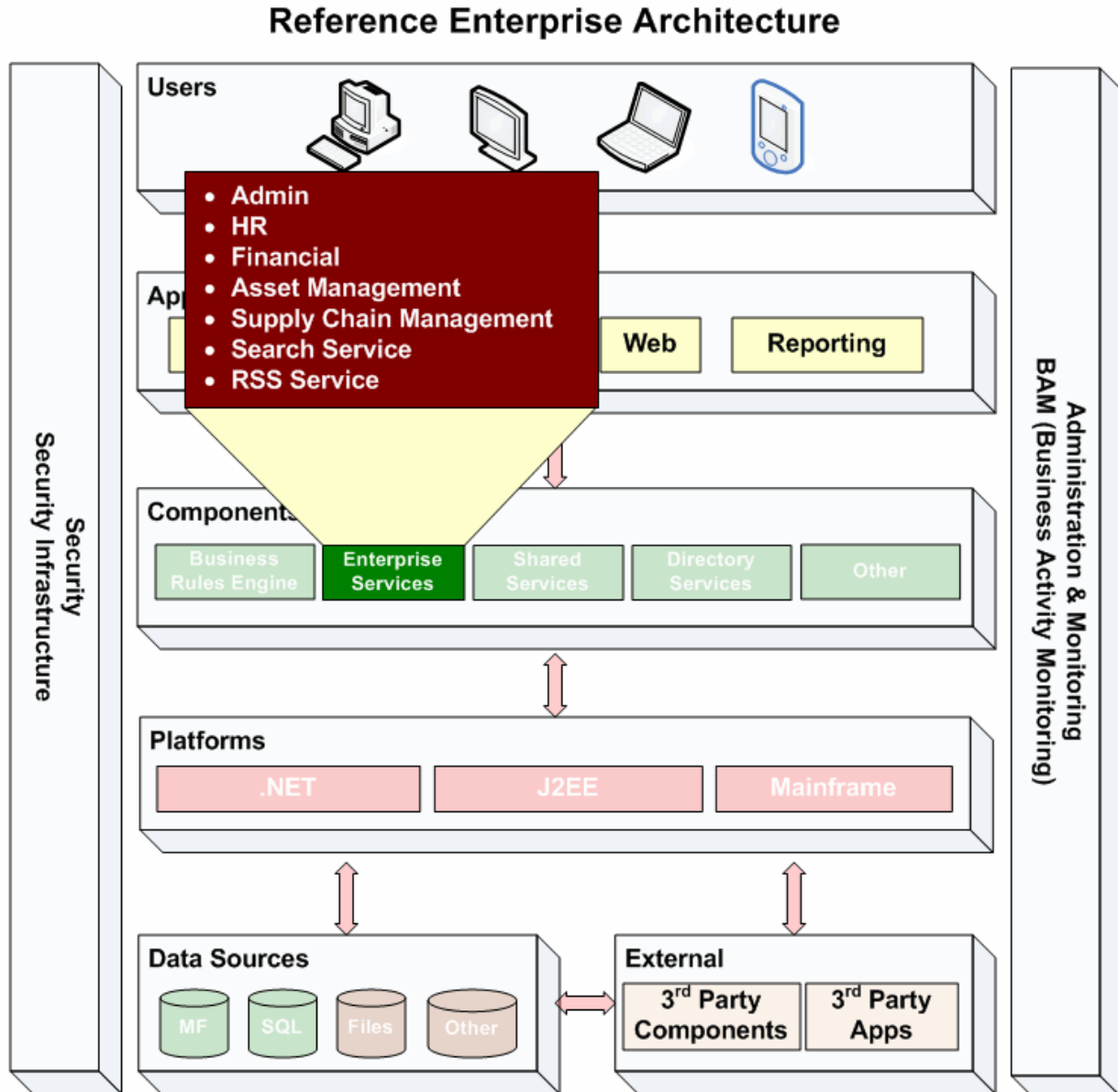
In most situations, applications should be moved to a browser-based environment. This simplifies client management and facilitates ease of use.



Enterprise Services

Business services in this category have state-wide scope. It is recommended that usage be mandatory since the most efficient usage of an enterprise service comes from using what has already been built. These services are typically provided via a COTS/packaged application. Examples are SAP and Oracle Applications for HR, Admin, Financial, Supply Chain and Asset management.

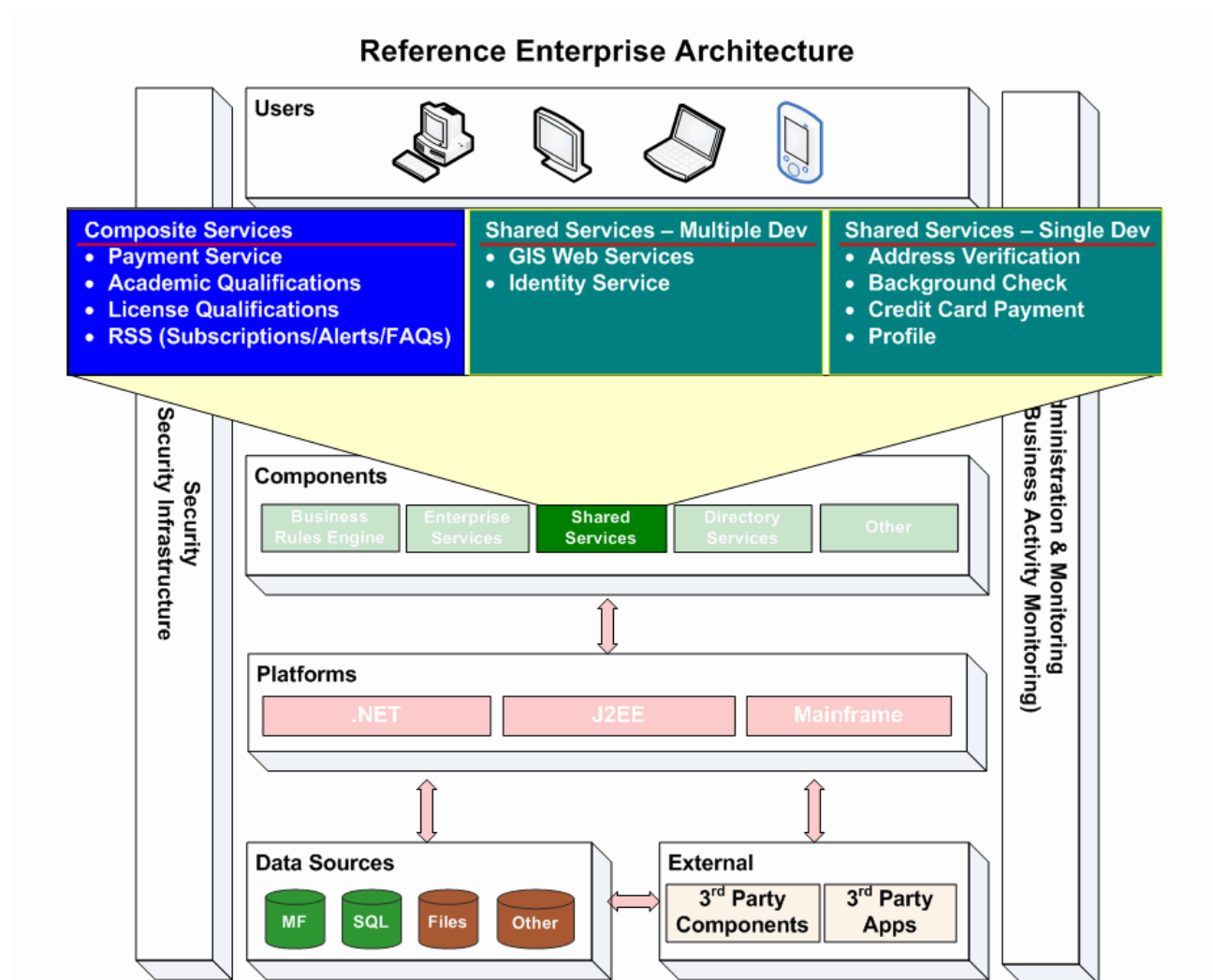
However, other types of applications also meet these criteria such as an enterprise search service and a RSS (Real Simple Syndication) service. The latter provides subscription, alert, news, and FAQ's general purpose services.



Shared Services

This class of service has a scope of “community of interest”. That is, they are not state-wide, but are very important to a particular group that share common business requirements. Shared services are usually not stand-alone; rather they are consumed by business applications. Therefore, they are ideally suited for a component-based architecture such as web services.

Shared services fall into three categories; those developed by a single organization, those that are primarily developed collaboratively by multiple organizations, and composite services. An Address Verification Service is a good example of a single development, while GIS Web Services is a great example of multiple organization development.



Composite services are an aggregate or “roll up” of base (“atomic”) services. These are typically orchestrated in a business process fashion. While there are competing standards in this area, it appears that BPEL (Business Process Execution Language) is the more popular one. So, it is important to pick a vendor tool that supports BPEL. A Composite Service example might be a Payment Service which could be a re-assembly of Credit Card Payment Service and EFT Payment Service.

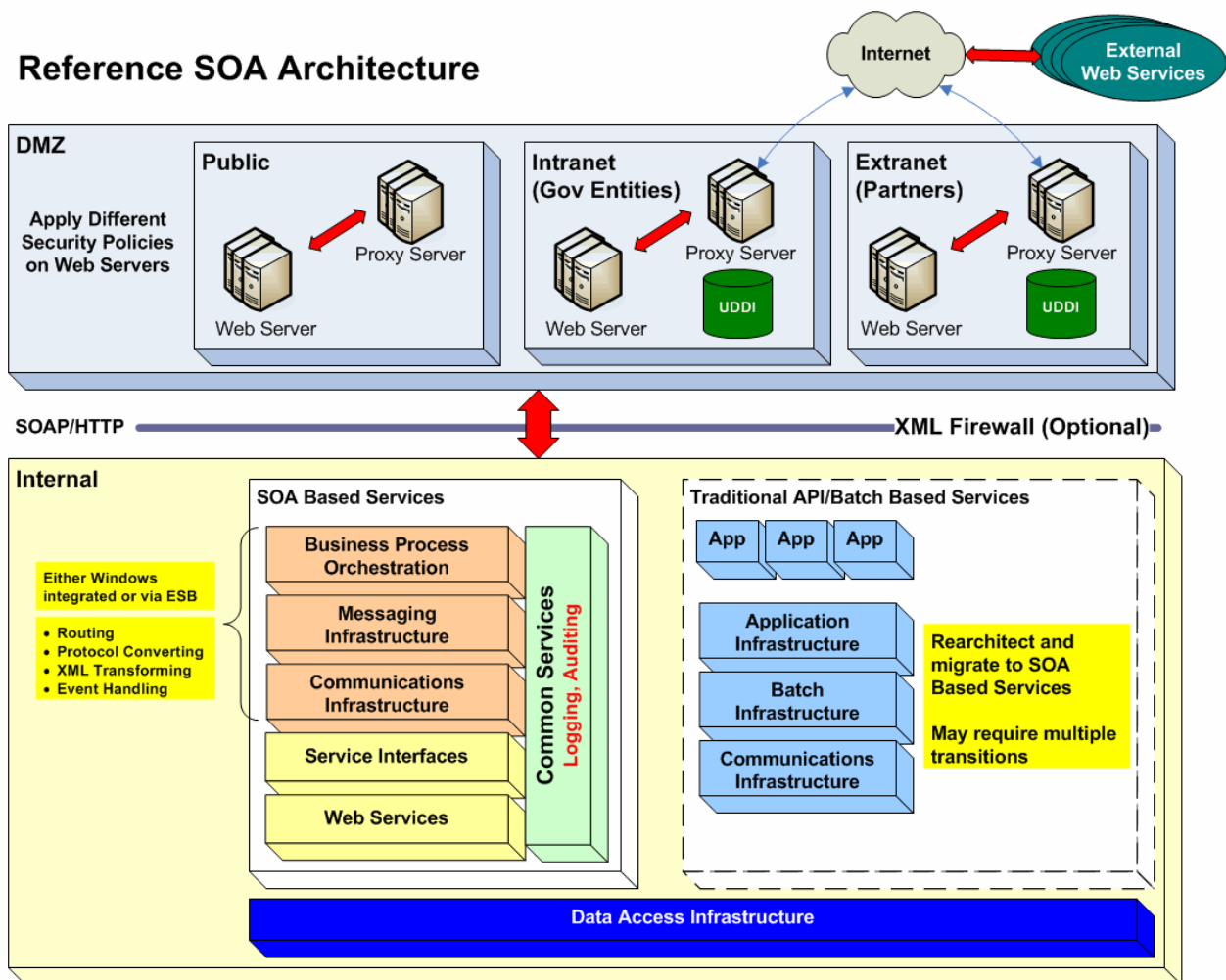
Because of the distributed nature of shared services, they need a good management environment. A tool will be required to manage the orchestration of web services, handle XML messaging that web services use to

communicate, is based on open standards, provides mediation between services, coordinates interactions and ensures reliable delivery of the messages.

Reference SOA Architecture

Establishing an Enterprise Reference Architecture is important for the big picture. SOA is a key subset of the enterprise and it is sometimes not obvious where SOA fits into the enterprise. That is, one can get lost in the many details and standards surrounding SOA. So, a Reference SOA Architecture is provided.

Note web services can be either internal or external to an organization. Using services developed and made public by other organizations is highly encouraged to reduce duplication of resources.



From a security perspective, it is desirable to put as much in the Internal tier as possible. Only components located in the DMZ are accessible via the Internet. The DMZ could be architected to provide different levels of security based on profile group. Proxy services and security policies could be applied at the web server level.

In the above diagram, Traditional API/Batch Based Services are contrasted to SOA Based Services. The traditional environment reflects current “stove-pipe” applications and their complex (and duplicative)

infrastructure. It is noted that these applications should have a retirement strategy which includes migration details. In many cases, multiple phased projects will be required.

In the SOA Based Services perspective, Web applications manage user interactions, and they invoke services that implement business processes. Web applications invoke web service APIs via XML interfaces which are message based. A proxy mirrors the actual web service interface. Web services are defined in a WSDL (Web Services Description Language) document which may be registered in a UDDI repository (which provides location services).

Services should be implemented in the Internal tier. The XML messages are processed by the messaging infrastructure when the appropriate service is called by a business process. An XML Firewall could be deployed to look inside SOAP messages and enforce the security section of the message. Web services are implemented as a Business Component in a specific language (.NET/C# or J2EE/Java). Access Services handle formatting and communications among data sources including packaged applications, rules, report, and security servers.

This document does not address the complex issue of data management. This is a large topic which needs proper attention to full realize the potential of an SOA Based Services environment.

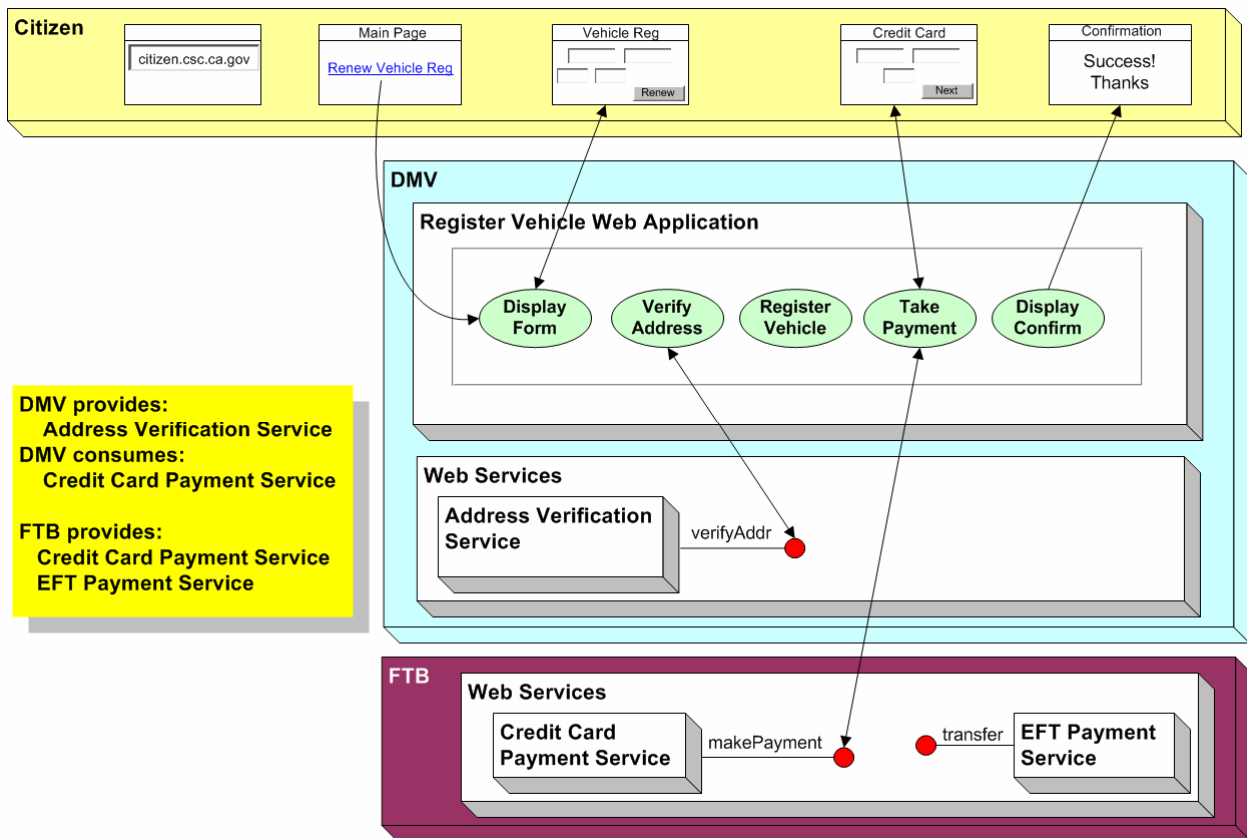
SOA Service Patterns

There are a number of different ways in which web services can be used. Following, are patterns that illustrate a few of the many patterns used by web service architects. The patterns are described using example scenarios to help clarify how one might use the pattern.

The first pattern is the most basic. It illustrates how consumers and producers work in an SOA environment. The other patterns focus on more complex federated and enterprise issues. The patterns are intended to paint a picture and are not intended to represent complete models.

Application Consuming Web Services Pattern

A user is directed to the Register Vehicle Web Application at DMV. Function modules within this application directly invoke (consume) Address Verification and Credit Card Payment web services. FTB is a service *provider* with the Credit Card Payment and EFT Payment services. DMV is both a *provider* and *consumer* of the Address Verification service.



Consuming Web Services Pattern

Other web applications might consume Address Verification, Credit Card Payment, and EFT Payment services. In fact, they would be registered with a UDDI repository and available to any application (with appropriate policy and security credentials).

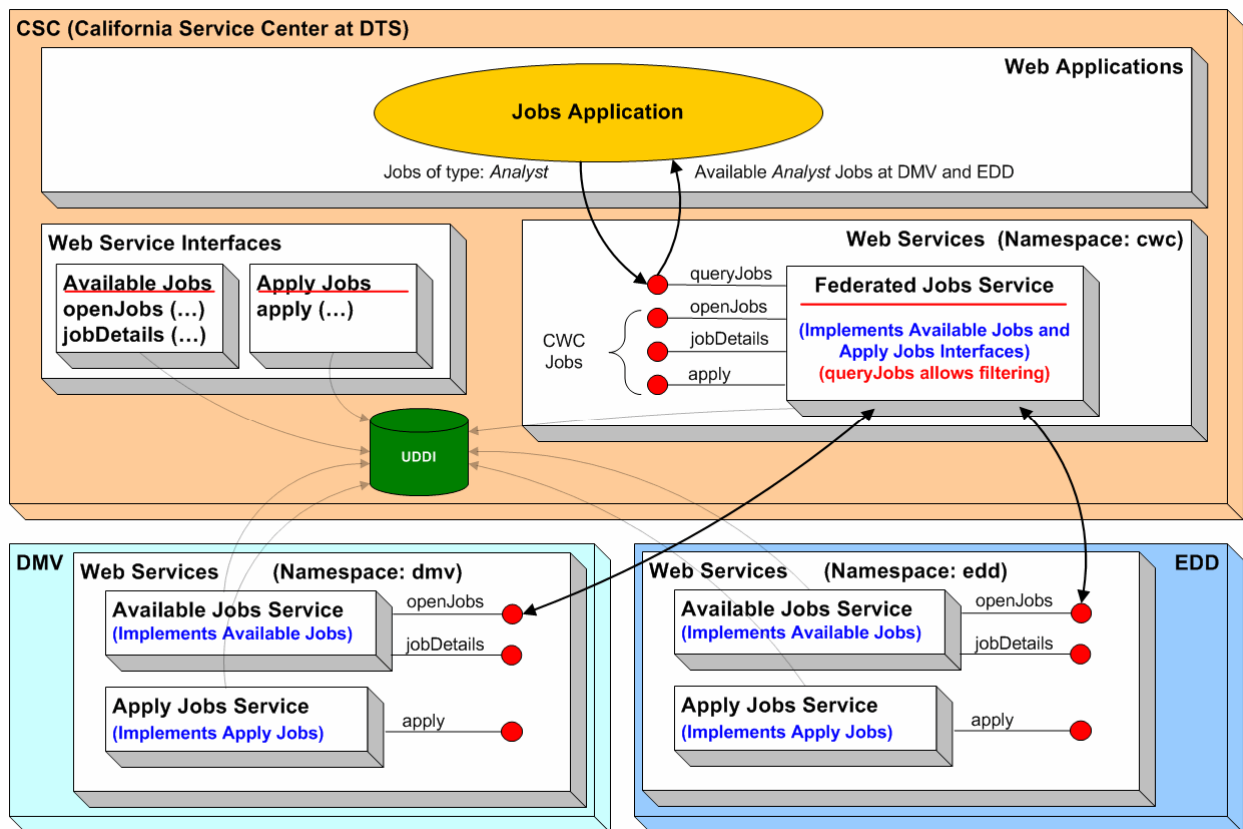
The fact that the vehicle registration application is using web services from multiple providers is completely hidden from the user as these enterprise components collaborate behind the scenes to fulfill this business service request. The customer experience is a seamless end-to-end business interaction regardless of the collaboration efforts required behind the scene.

Federated Service Interfaces Pattern

In many cases it makes sense for the CSC to handle user interactions as a single, customer-centric site. For example, a web application might be developed and run at CSC that would serve as a single, easy to use and always up to date place for all state employees to go for available jobs. In the below example, the actual departments would supply the most current job details while the CSC application would handle filtering and presenting the information in a user-friendly fashion. The displayed information could be organized by department, or by job classification, and not only provide available job details but also handle applying for a job.

This example demonstrates the use of web service *interfaces*. An Available Jobs interface is created with two web methods (openJobs and jobDetails). An Apply Jobs interface is also created. These interfaces (along with their identifiers and XML schemas) are placed in the UDDI repository and made available for use in any web service. CSC, DMV, and EDD have each created an Available Jobs Service which implements these interfaces. Since the service name is the same, each department must have a different

namespace. This enforces standardized behavior while allowing different content. In this case, the Available Jobs Service at DMV only returns DMV jobs.



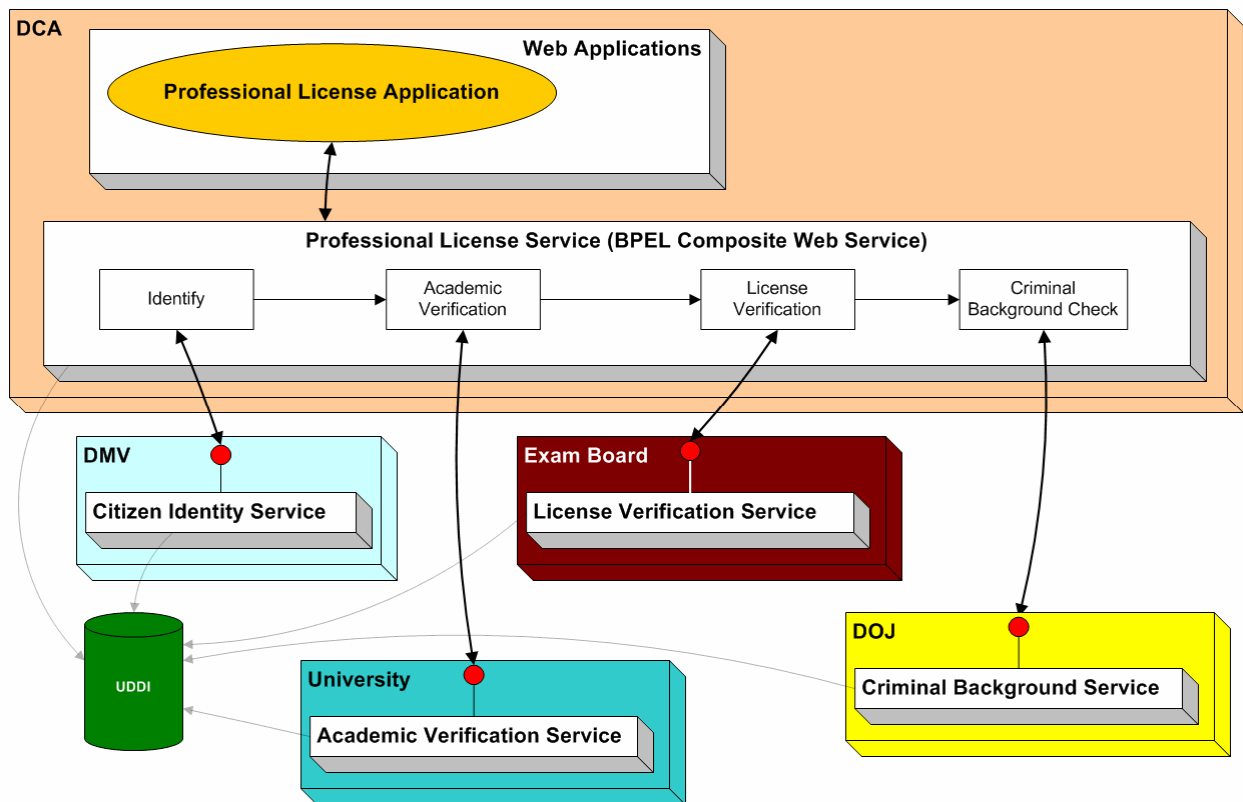
Federated Web Services Pattern

The above example also demonstrates the notion of web service federation. Federated Jobs Service implements a web method called `queryJobs` which is invoked by the Jobs Application. Federated Jobs Service then consumes Available Jobs Services at both DMV and EDD. So, the content returned to Jobs Application is federated among the web services. Federated Jobs Service might have additional logic for filtering jobs and categorizing them across departments.

Federated Composite Web Services Pattern

Web services can be aggregated to form higher level, or more coarse-grained services. The previous example demonstrated one web service (Federated Jobs Service) directly consuming other web services (Available Jobs Service at DMV, Available Jobs Service at EDD). An alternative would be to use a *composite* web service.

In the following example, Professional License Service is a composite web service that is compliant with the BPEL standard. This means one can orchestrate the flow of consumed web services. In this example, the Identity node consumes Citizen Identity Service at DMV. The Academic Verification node consumes Academic Verification Service at a university. The License Verification node combines the License Verification Service at an exam board. Finally, the Criminal Background Check node consumes Criminal Background Service at DOJ.



Composite Web Services Pattern

One might use a composite web service when there are many services to be consumed. The order in which they are consumed can be easily changed.

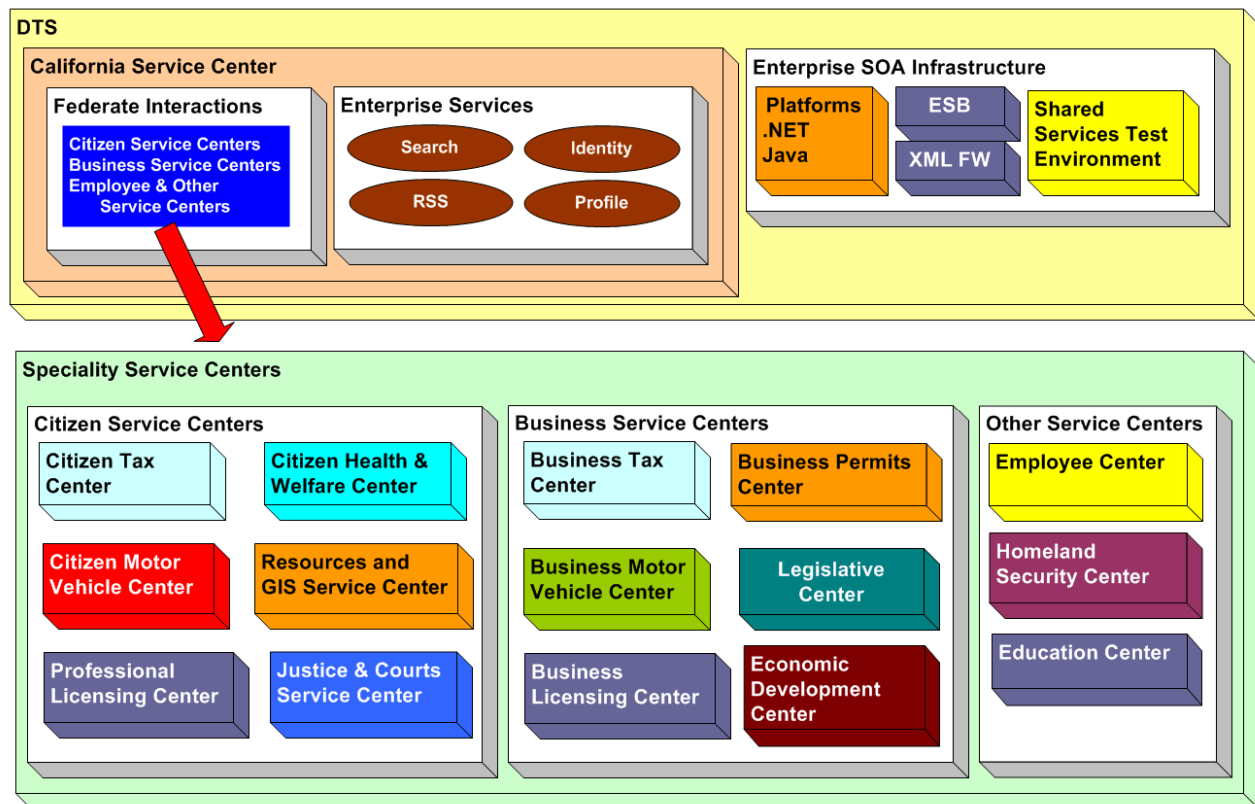
Federated Service Centers Pattern

Ideally, California should appear as one entity from the constituent perspective. There are basically four ways constituents will access California services:

1. Via the new California Service Center (CSC) at DTS.
2. Via industry search engines such as Google, Yahoo, or MSN.
3. Via other government web sites such as FirstGov.
4. Access a Specialty Service Center (SSC) directly.

For those users that come through CSC, the interaction might be fulfilled by one of the enterprise services provided by CSC. However, in many cases the interactions will need to be federated to a Specialty Service Center. To make this even more convenient the users can optionally provide a profile so the content on their initial page at CSC is tailored to their areas of interest.

The overall goal is to minimize duplication of content. SSC's should focus on providing content that is specific to their line of business. Commonality among service centers should be factored into an enterprise service and deployed at CSC. An SSC can redirect to another SSC to seamlessly interact with the user.



There should be a standard look and feel state wide across all service centers. That is navigation and menu layouts should look the same and be located in the same area on pages. This dramatically improves the user experience. Where common, button and links should use the exact same names which make the applications more intuitive. Special note, the task of Information Architecture as it relates to navigation, look and feel is currently assigned to the California Research Bureau.

Note, one of the enterprise services offered by CSC is Identity service. This service is responsible for determining if security is required and, if so invoke the appropriate Identity Authority. However, since a user could go directly to an SSC, they also need to provide an Identity Service which federates to the same Identity Authorities.

Also note there should only be one RSS Service state-wide. Each department would individually keep their RSS file updated which the centralized RSS Service at CSC would handle.

Enterprise Search Service Pattern

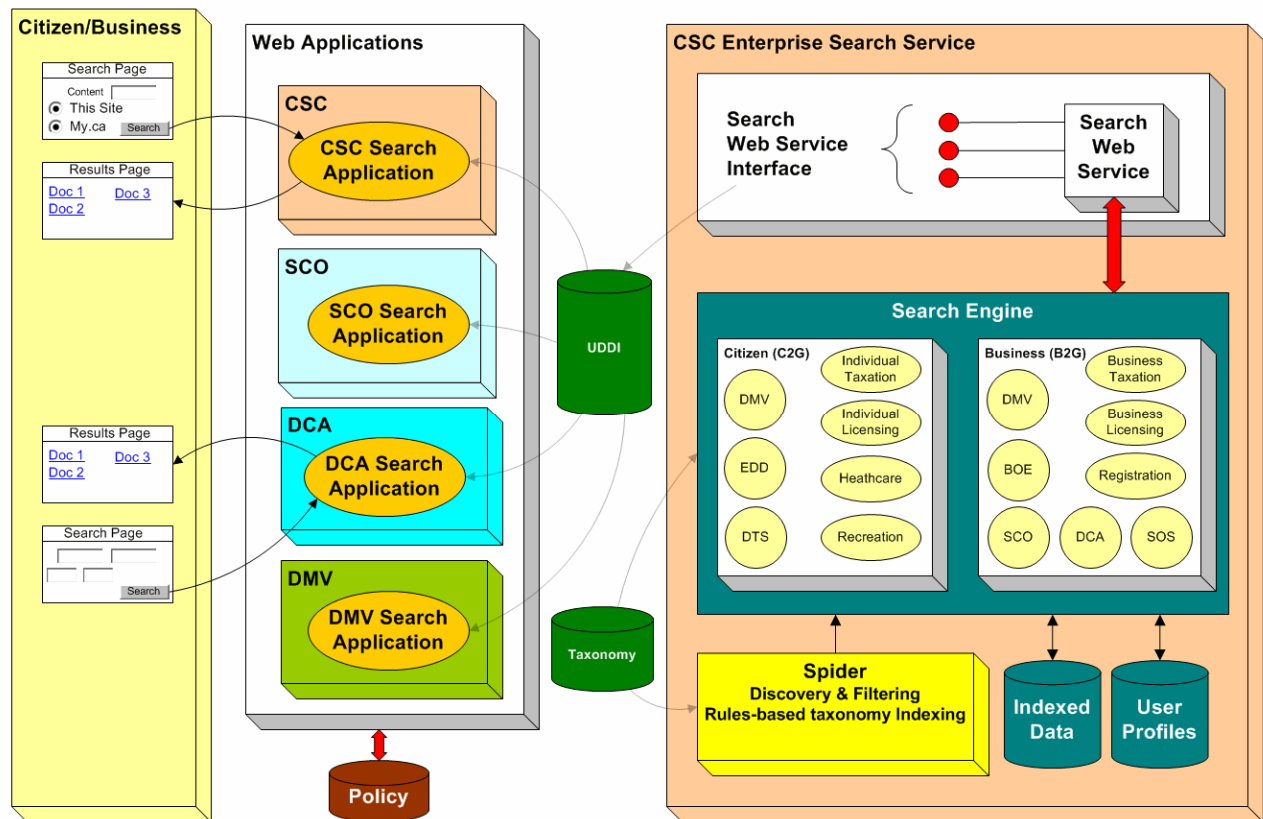
The Enterprise Search Service is a more complex example of federated services. The current State search engine could be redesigned to handle a variety of different search requests while returning more meaningful results. That is, results that meet customer expectations more closely.

In the next example, a web services interface could be utilized in conjunction with a search engine. This interface would expose the types of searches that any department application could implement. So, consistency across departments would be achieved as it would appear from the user perspective as if they were interacting with a single search application.

In fact, the search engine could be configured to index information in a more useful manner such as by user type (citizen vs business). Additionally, filtering could be applied to the spider process to not index certain content (for example, those without titles, or those defined in duplicate locations).

If the state came up with a common language (taxonomy) for defining content, then the results of the spider could be indexed according to the taxonomy.

This is a federated services approach since the indexes are maintained at DTS, but the actual content is located in each department.



Enterprise Search Service Pattern

Search applications could use profile and policy information to influence search results. Additionally, search applications can use different parts of the search interface, as well as aggregate or filter the results. So, the user experience could be very different. A search application could ask the search engine (via the search interface and broker) to only look only in certain collections or to use only certain parts of a taxonomy.

Federated Search Engines Pattern

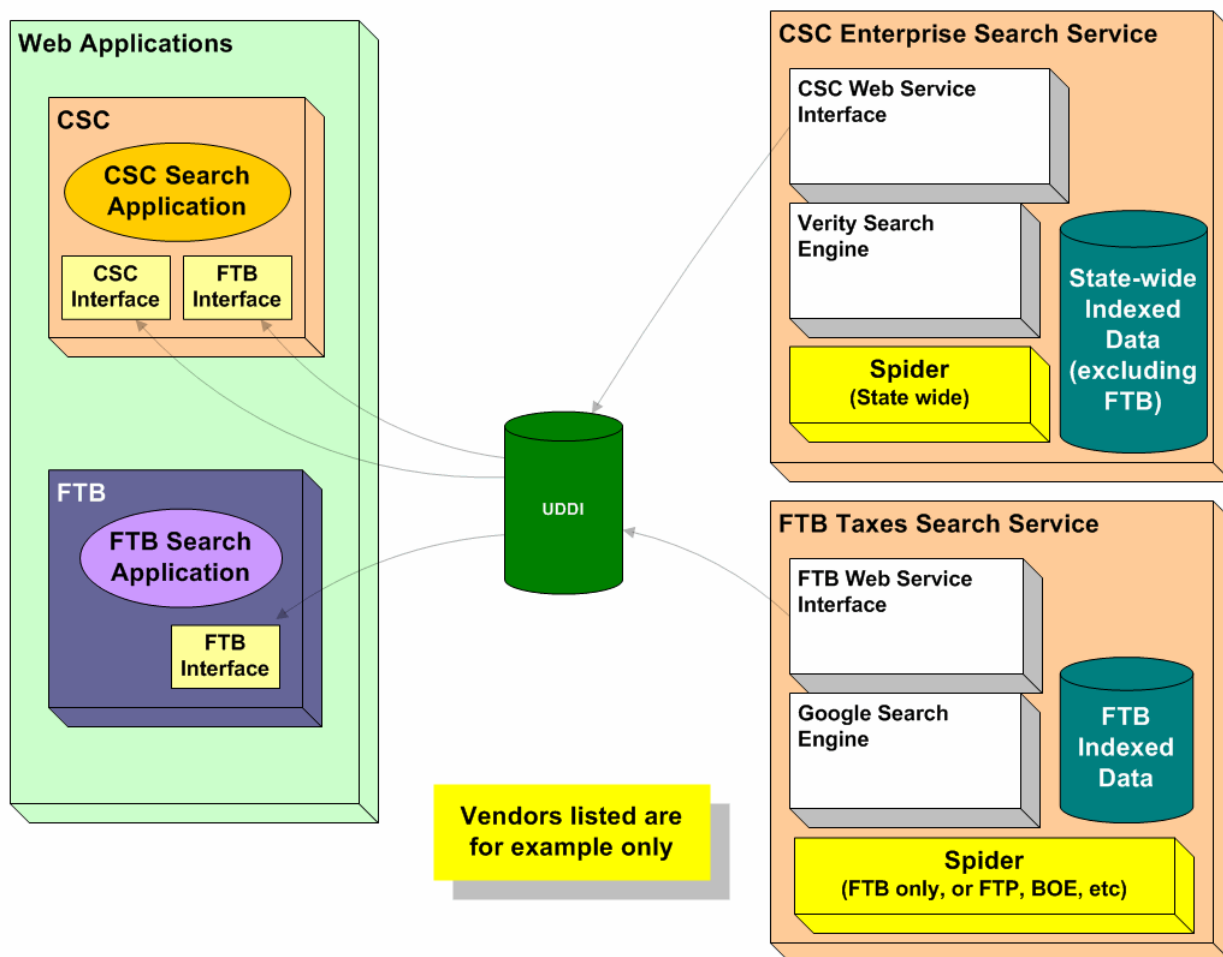
The above example illustrates a centralized search engine approach. Economies of scale, lower infrastructure costs, consolidated databases, lower spider traffic, and lower staff costs are some of the reasons to use this approach. It is particularly effective as a general state-wide search service that, if architected properly could intelligently defer to topic specific department search engines.

Departments maintain knowledge and expertise that could be captured in their own search engines. For example, it might make sense for Franchise Tax Board to maintain a Tax Search Engine where one could go

to get answers to any tax question. DHS might maintain a Health Search Engine, DOJ a Criminal & Justice Search Engine, etc.

Depending on the scope of the Tax Search engine, FTP might spider just their own site or they could spider FTP, BOE, and other sites that deal with taxes. It is likely that this search engine would return a more comprehensive result set than the state-wide search engine.

FTB would define a web service interface for their search engine and register this interface with the UDDI repository. Then, any search application would have the choice of implementing the CSC or FTB search interface (or both interfaces).



Federated Search Engines Pattern

In the above diagram, CSC could implement both interfaces, and based on the users query CSC could federate to FTB if the user is looking for tax information. If both search engines are from the same vendor, this capability may be built in. That is, logic to determine which search engine is in the best position to service the query is determined by search engine collaboration.

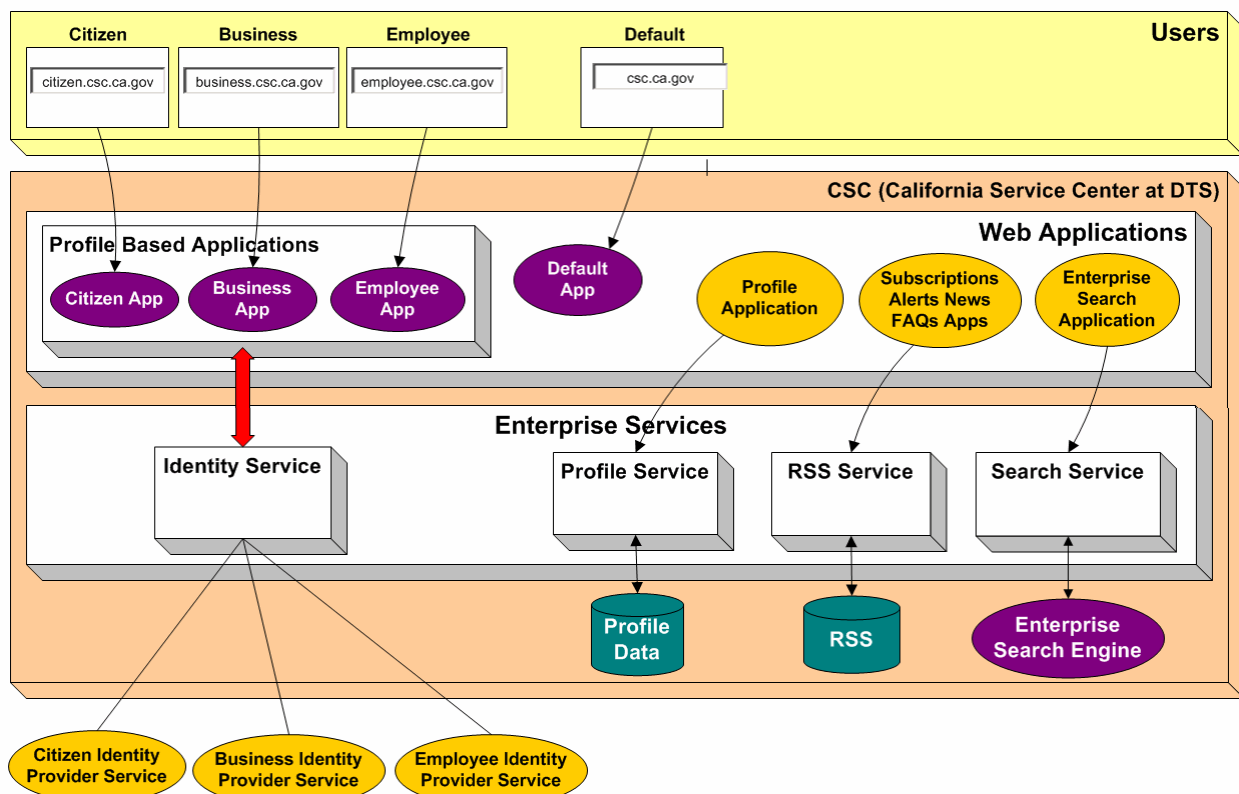
California Service Center (CSC) at DTS Enterprise Service Examples

The current State Portal will be replaced by the California Service Center (CSC). Its goal will be to make it the primary point of entry for all users interacting with California. The California Service Center will be a key customer-facing part of the new services oriented enterprise environment.

The CSC could display a default page which would organize information for all users in an intuitive manner. In many cases, it would intelligently redirect users to a specialty service center. The CSC may be organized by user service type, for example Citizen, Business, Employee, etc. This means the user screens could be tailored based on the type of user. That is, all information of interest to Citizens could be grouped into a Citizen main page. This implies that the CSC is enabled to identify the difference between user types. In many cases this could be handled via profiles which would allow CSC to automatically return the correct page type. Another way might be to let the user select which profile they want for a given session. In any case, specific domain names could be established to differentiate user types. For example, csc.ca.gov could resolve to the main CSC page where a user could select a profile, citizen.csc.ca.gov would return the main page for Citizens, business.csc.ca.gov for Business uses, etc.

There are certain enterprise services that are good candidates to run at CSC. Links and buttons on any page could be encoded to pass whether or not identification is required and if so, what level. The CSC application could then invoke an Identity Service which would communicate with an “identity provider”. See [California SOA Security Model](#) for more details. Once the user is identified by the appropriate provider service, the request could be handed to the appropriate application along with the security token.

Another CSC service example might be a profile application and a Profile Service to retrieve customer preferences. These preferences might be used to alter the page layout and content as well as for personalization. The Profile Service could be consumed by other applications, anywhere profile information is needed.



California Service Center – Enterprise Services

RSS is another great enterprise service for CSC to offer. See next section for details.

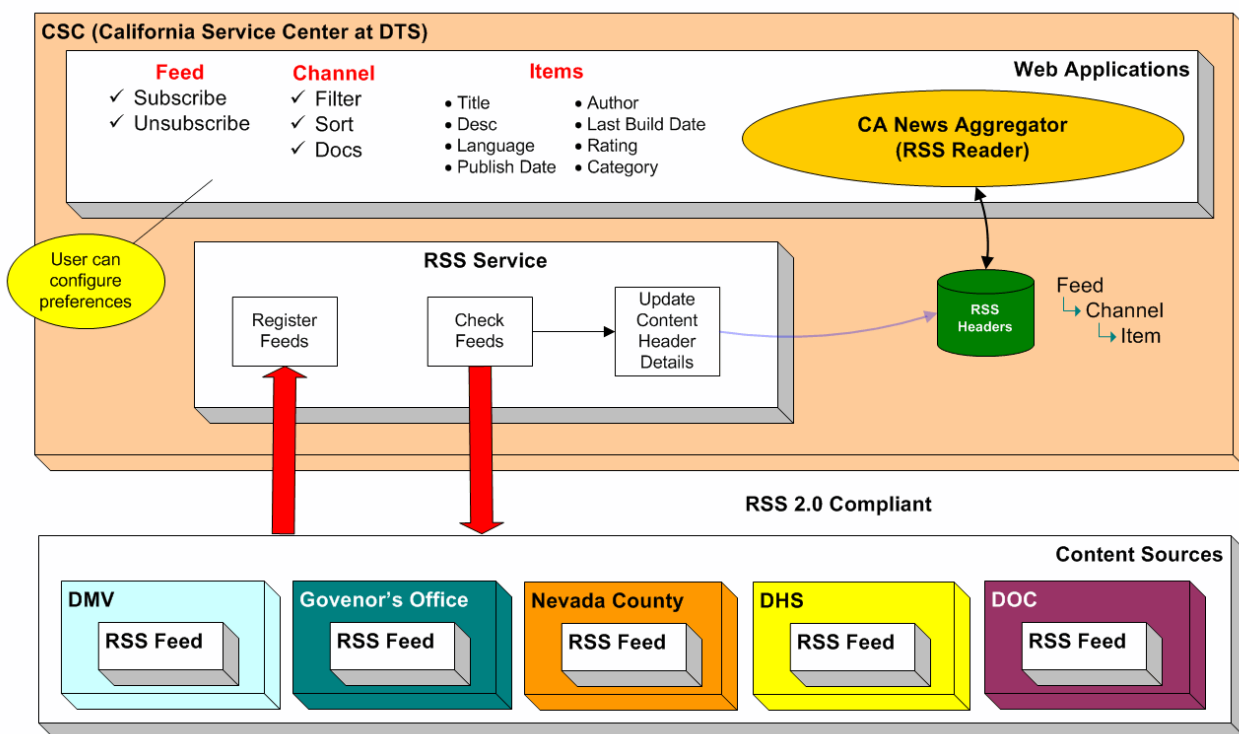
RSS (Real Simple Syndication) Pattern

RSS is a very popular method of publishing regularly updated information. It was originally created by Netscape and used primarily by news and media companies as a way of syndicating news. When Netscape lost interest, Userland Software picked it up and continued to enhance it. A parallel effort using a different format was initiated by the W3C standards body. Fortunately, the formats have merged with RSS 2.0. It is now widely used by many companies and public entities as an easy way to keep customers informed of changes.

A couple of public sector examples are the states of Virginia and Utah. Virginia provides 34 different feeds on topics like Featured Sites, Emergency Notifications, Press Releases, Citizen Services, Online Services, Family Services, Educational Services, Government Resources, License and Permits, Forms, and Business Services. Changes to any of the above topics are placed in the appropriate RSS file which is registered with organizations that monitor RSS feeds.

Anyone can download a free RSS Reader (called a News Aggregator), and subscribe to any RSS feed directly or to any of the many published sites that allow an individual to subscribe to multiple feeds. Additionally, RSS information can be pushed out to cell phones and PDA devices.

The California Service Center could provide an RSS Service that monitors the feeds from all state departments. Then, anyone could subscribe to the service and pick which feeds they were interested in and what format they would like to receive the information.



From a technical perspective, open source class libraries already exist that do most of the work. Some examples: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnexxml/html/xml02172003.asp>
<http://www.rssdotnet.com/>
<http://aspnet.4guysfromrolla.com/articles/102903-1.aspx>

In addition to the above Microsoft-based components, there are also many freely available products that are written in PERL, PHP, JavaScript, and Java.

Here are links to states:

Virginia <http://www.rssgov.com/archives/000127.html>

Utah <http://www.utah.gov/>

Utah sponsors an RSS workshop <http://www.rssgov.com/rssworkshop.html>. This site has a great list of other state RSS feeds as well as a good, detailed explanation of RSS.

Any site that displays either **RSS** or **XML** publishes an RSS feed.

California Business Reference and Service Reference Models

In a service-oriented architecture, all business services are defined in the business reference model (BRM). The BRM is part of the Enterprise Repository (core.ca.gov). One of the key principles behind SOA is to break down business services into reusable components that can be combined and shared across the enterprise. These shared components are called web services and they are defined in the service reference model (SRM) which is also located in the Enterprise Repository.

Both the BRM and SRM are hierarchical. The exact structure of the model will be determined at design time. However, for the purpose of illustration, the following diagram shows one way the BRM and SRM might be organized.

Business Reference Model					Service Reference Model			
Customer Audience	Business Service Group	Line Of Business	Business Function	Business Service	Service Domains	Service Types	Service Components	WS Type
C2G	Regulatory & Compliance	Licensing	Professional Licensing	Medical Doctor License	Business Management Services	Payment Services	Credit Card Payment Service	Base
						Customer Services	Address Verification Service	Base
					Authorization Services	Professional License Qualifying Services	Check Criminal Background Service	Base
							Check License Qualifications Service	Base
							Check Qualifications Fulfillment Service	Base
C2G	Financial Assistance	Title IV Grants	Post-Secondary Education	Cal-Grant	Business Management Services	Payment Services	EFT Payment Service	Base
					Grants Service	Grant Eligibility Services	Student Financial Eligibility Service	Base
							Student Academic Eligibility Service	Base
B2G	Revenue Collection	Business Tax Payments	Employer Income Taxes	Personal Income Tax	Business Management	Payment Services	Business Payment Service	Composite
				State Disability Tax	Reporting Services	Employer Reporting Services	Base Wage Reporting Service	Base
B2G	Regulatory & Compliance	Licensing	Permits	Encroachment Permit	Business Management Services	Payment Services	EFT Payment Service	Base
							Credit Card Payment Service	Base
					Electronic Delivery Services	Issuance Services	Issue Permit Service	Base
						Confirmation Services	Email Confirmation Service	Base
E2G	Government Services Management	HR Management	Organization & Position Compensation Management	Position Control	Employee Services	Position Tracking	Personnel Transaction	Base
				Employee History		Emp Pos Track		
				Salary & Leave		Comp Tracking		
				Time & Attendance		Attend Tracking		

BRM: For a given customer audience (C2G, B2G, E2G, etc.), the top level category might be *Business Service Group*. This is decomposed into *Lines of Business*, which in turn are broken down into *Business Functions*, and finally, into actual *Business Services*. So, in the diagram example, Medical Doctor License might be a Business Service that is part of Professional Licensing Business Function which is part of the Licensing Line of Business, which belongs to the Regulatory & Compliance Business Service Group.

SRM: The SRM might have a structure of *Service Domains*, *Service Types*, and *Service Components*. So, in the above example, a Medical Doctor License Business Service might consume Credit Card Payment, Address Verification, Check Criminal Background, Check License Qualifications, Check Qualifications Fulfillment web services.

As an example of service reuse, the above diagram shows Credit Card Payment Service being used by Medical Doctor License and Encroachment Permit business Services. Notice, that Personal Income Tax Business Service is using Business Payment Service. This is a composite type of web service meaning that it includes other types of payment services (possibly, Credit Card Payment, EFT Payment, and Cash).

California SOA Security Model

Introduction

Exchange of information over the Internet is vital but may have security implications. Security issues over the Internet are important, because it is an insecure and untrustworthy public network infrastructure, prone to malicious attacks by professional and amateur intruders.

All of the information available for access over the Internet does not have the same level of business confidentiality. In the public sector, much information is intended to be accessed and viewed by anyone. However, there are a number of business transactions that do require knowing who the party is as well as the party's access privileges.

Organizations usually secure company resources available on the network and online services by defining business roles, access rights, and system policies. That's where firewalls, among other mechanisms, have a role to play. A network level firewall sits at the doorstep of a private network as a guard and typically provides the following security services:

- monitors all incoming traffic;
- checks the identity of information requesters trying to access specific company resources;
- authenticates users based on their identities, which can be the network addresses of service requesters or security tokens;
- checks security and business policies to filter access requests and verify whether the service requester has the right to access the intended resource; and
- provides for encrypted messages so that confidential business information can be sent across the untrustworthy Internet privately.

The main purpose of a firewall is to protect the physical boundaries of a network. There is a physical boundary of the private network and the only way to get into the network is through the firewall. While packets of network traffic and messages pass through a firewall, they are authenticated and checked for possible intrusion or malicious attacks.

When one department application interacts with an enterprise component provided by a different department, it cannot control and may not even know much about the IT infrastructure of the other department. For example, the first department might be using a Java solution over Solaris servers and the second department may be using .NET or some other technology. How can interoperability between the two departments be ensured?

It is not feasible to sit down with all departments and decide about messaging formats for exchange of information and interoperability. This will create an endless task of designing and redesigning message formats for each department. The cost of this type of legacy integration is so high that such techniques are only feasible in high IT return private sectors, such as airline and banking industries. Most government e-commerce isn't able to justify IT infrastructure development costs in this way.

But if applications shift from legacy integration to the Service Oriented Architecture (SOA) provided by web services, interoperability issues are much easier to deal with. A web services-based SOA depends on SOAP servers to process messages. A SOAP server holds only the information related to the web services it is hosting (names of the services, names of the methods in each service, where to find the actual classes that implement the web services, and so on) and has the capability of processing incoming SOAP requests. However, the SOAP server itself doesn't have any capability to check whether the incoming SOAP request is coming from an anonymous customer or a known business partner. SOAP cannot distinguish between sensitive and non-sensitive web services and cannot perform user authentication, authorization, and access control.

It is clear that a remote client who has accessed a SOAP server enjoys the opportunity of invoking any method of any services hosted on the particular SOAP server. So, one might correctly conclude that it is not safe to host web services of different levels of sensitivity on the same SOAP server.

Even if you deploy a network-level firewall to protect from intruders, you will not be able to distinguish between different users once it has reached the SOAP server. It is possible that an intruder authenticates himself as an anonymous user, reaches the SOAP server, and invokes sensitive web services meant for a different user. Thus, a SOAP server is like a hole in your network.

There are two solutions to this problem:

1. Use a different SOAP server for each level of sensitivity, so that different authentication policies can be enforced on each sensitivity level. This solution may seem appropriate for web services today. However the real advantage of web services lies in the next generations where web services will not just be invoked by browser-assisted human clients, but web services will invoke each other to form chained or transactional operations. Such complex web service infrastructure will be very hard and expensive to build with the idea of having a separate SOAP server for each authorization policy. In addition, this idea hardly allows building reusable or off-the-shelf security solutions.
2. The second option is to make the firewall XML and SOAP aware. The firewall will be able to inspect SOAP messages, trying to match user roles with access lists, policy levels, and so on. This solution is a better approach. It also allows building XML-based standard security protocols, which can be adopted by security vendors to ensure interoperability.

3.

Web service users can add security information (signature, security tokens, and algorithm names) inside SOAP messages, according to the XML-based security protocols. The XML and SOAP-aware firewall will check the message before it reaches the SOAP server, so that it is able to detect and stop intruders before they are able to reach the service.

Based on the second approach described above, W3C and OASIS are developing several XML-based security protocols. These protocols will define the various security features of an XML and SOAP-aware firewall.

XML Security for Web Services

This section very briefly discusses the high level features of some of the security protocols from W3C and OASIS.

The *XML Signature* specification is a joint effort of W3C and IETF. It aims to provide data integrity and authentication (both message and signer authentication) features, wrapped inside XML format.

W3C's *XML Encryption* specification addresses the issue of data confidentiality using encryption techniques. Encrypted data is wrapped inside XML tags defined by the XML Encryption specification.

WS-Security from OASIS defines the mechanism for including integrity, confidentiality, and single message authentication features within a SOAP message. WS-Security makes use of the XML Signature and XML Encryption specifications and defines how to include digital signatures, message digests, and encrypted data in a SOAP message.

Security Assertion Markup Language (SAML) from OASIS provides a means for partner applications to share user authentication and authorization information. This is essentially the single sign-on (SSO) feature being offered by all major vendors in their e-commerce products. In the absence of any standard protocol on sharing authentication information, vendors normally use cookies in HTTP communication to implement SSO. With the advent of SAML, this same data can be wrapped inside XML in a standard way, so that cookies are not needed and interoperable SSO can be achieved.

eXtensible Access Control Markup Language (XACML) presented by OASIS lets you express your authorization and access policies in XML. XACML defines a vocabulary to specify subjects, rights, objects, and conditions -- the essential bits of all authorization policies in today's e-commerce applications.

Basic Cryptographic Concepts

The discussion of message integrity, user authentication, and confidentiality employs some core concepts: keys, cryptography, signatures, and certificates. Following, cryptographic basics will be briefly discussed.

Asymmetric cryptography

A popular cryptographic technique is to use a pair of keys consisting of a public and a private key. First, you use a suitable cryptographic algorithm to generate your public-private key pair. Your public key will be open for use by anyone who wishes to securely communicate with you. You keep your private key confidential and do not give it to anybody. The public key is used to encrypt messages, while the matching private key is used to decrypt them.

In order to send you a confidential message, a person may ask for your public key. He encrypts the message using your public key and sends the encrypted message to you. You use your private key to decrypt the message. No one else will be able to decrypt the message, provided you have kept your private key confidential. This is known as *asymmetric encryption*. Public-private key pairs are also sometimes known as *asymmetric keys*.

Symmetric cryptography

There is another encryption method known as symmetric encryption. In symmetric encryption, you use the same key for encryption and decryption. In this case, the key has to be a shared secret between communication parties. The shared secret is referred to as a symmetric key. Symmetric encryption is computationally less expensive than compared to asymmetric encryption. This is why asymmetric encryption is ordinarily only used to exchange the shared secret. Once both parties know the shared secret, they can use symmetric encryption.

Message digests

Message digests are another concept used in secure communications over the Internet. Digest algorithms are like hashing functions: they consume (digest) data to calculate a hash value, called a message digest. The message digest depends upon the data as well as the digest algorithm. The digest value can be used to verify the integrity of a message; that is, to ensure that the data has not been altered while on its way from the sender to the receiver. The sender sends the message digest value with the message. On receipt of the message, the recipient repeats the digest calculation. If the message has been altered, the digest value will not match and the alteration will be detected.

But what if both the message and its digest value are altered? That kind of change may not be detectable at the recipient end. So a message digest algorithm alone is not enough to ensure message integrity. That's where we need digital signatures.

Digital signatures

Keys are also used to produce and verify digital signatures. You can use a digest algorithm to calculate the digest value of your message and then use your private key to produce a digital signature over the digest value. The recipient of the message first checks the integrity of the hash value by repeating the digest calculation. The recipient then uses your public key to verify the signature. If the digest value has been altered, the signature will not verify at the recipient end. If both the digest value and signature verification steps succeed, you can conclude the following two things:

- The message has not been altered after digest calculation (message integrity); and
- The message is really coming from the owner of the public key (user authentication).

Certificates

In its most basic form a digital certificate is a data structure that holds two bits of information: The identification (e.g. name, contact address, etc.) of the certificate owner (a person or an organization); and the public key of the certificate owner.

A certificate issuing authority issues certificates to people or organizations. The certificate includes the two essential bits of information, the owner's identity and public key. The certificate issuing authority will also sign the certificate using its own private key; anyone interested party can verify the integrity of the certificate by verifying the signature.

Message Integrity and User Authentication with XML Signatures

The XML Signature specification, XML Digital Signature, (XMLDS) has been jointly developed by W3C and IETF. It has been released as a recommendation by W3C. XML Signature defines the processing rules and syntax to wrap message integrity, message authentication, and user authentication data inside an XML format.

As an example, a department includes message integrity and user authentication information within a SOAP method invocation. The XML firewall of the department receiving the message, on receipt of the invocation, will need to look into the SOAP message to verify that:

- The message has not been altered while on its way to the web service (message integrity); and
- The requester is really the trusted user (user authentication).

The XML firewall will only let the request pass onto the SOAP server if both these conditions are met. It should be noted that XMLDS, isn't SOAP-specific. XMLDS can be used to insert signatures and message digests into any XML instance, SOAP or otherwise.

An XMLDS implementation can create SOAP headers to produce signed SOAP messages. The XML firewall sitting at the recipient's end will process the SOAP header to verify the signatures before forwarding

the request to the SOAP server. We can achieve the following two security objectives through this procedure:

- We can verify that the SOAP message that we received was really sent by the sender we think it came from.
- We can verify that the data we received has not been changed while on its way and is the same that the sender intended to send.

XML Encryption

The XML Encryption specification satisfies confidentiality requirements in XML messages. XML encryption offers several features.

- You can encrypt a complete XML file.
- You can encrypt any single element of an XML file.
- You can encrypt only the contents of an XML element.
- You can encrypt non-XML data (e.g. a JPG image).
- You can encrypt an already encrypted element (i.e., "super-encryption").

XML Encryption Processing

How will our XML firewall work with these encryption concepts? It will receive SOAP messages with encrypted elements or content and translate the contents to a decrypted form before forwarding the decrypted SOAP message request to the SOAP server.

The recipient of an XML encrypted file will decrypt the XML encrypted file in the following sequence:

1. Extract the encrypted content of the CypherValue element.
2. Read the algorithm attribute value of the EncryptionMethod element.
3. Read the Type attribute values of the EncryptedData element.
4. Obtain the keying information from the ds:KeyInfo element.
5. Use the information gathered in steps 1, 2, 3, and 4 to construct the plain text (decrypted) file.

An Introduction to Web Service Security

How will our XML firewall use XML signatures and encryption to protect SOAP servers? We have seen examples of using the two technologies individually, but the question of how to apply these two technologies in an XML firewall application to protect a SOAP server still needs to be addressed, especially since neither XMLDS nor XML Encryption are SOAP-specific. So why have we put all the signature related information in the SOAP header? Why not wrap it inside the SOAP body?

The Web Services Security (WSS) specification from OASIS defines the details of how to apply XML signature and XML encryption concepts in SOAP messaging. WSS relies on XMLDS and XML encryption for low level details and defines a higher-level syntax to wrap security information inside SOAP messages.

WSS describes a mechanism for securely exchanging SOAP messages. It provides the following three main security features:

1. Message Integrity
2. User Authentication
3. Confidentiality

It is an example SOAP message that carries security information according to the WSS syntax. Notice the request's header is carrying digital signature information.

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP:Envelope
  xmlns:SOAP="http://www.w3.org/2001/12/soap-envelope"
  xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/seceext"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <SOAP:Header>
    <wsse:Security>
      <wsse:BinarySecurityToken
        ValueType="wsse:X509v3"
        EncodingType="wsse:Base64Binary"
        wsu:Id="MyCertificate">
        LKSAJDFLKASJDLkjlkj243kj;lkjLKJ...
      </wsse:BinarySecurityToken>
      <ds:Signature>
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
          <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
          <ds:Reference URI="#myRequestBody">
            <ds:Transforms>
              <ds:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            </ds:Transforms>
            <ds:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <ds:DigestValue>BSDFHJYK21f...</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>
        GKLLKAJFLASKJ52kjKJKLJ345KKKJ...
        </ds:SignatureValue>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#MyCertificate" />
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </SOAP:Header>
  <SOAP-ENV:Body>
    <s:GetMyAccountBalances
      xmlns:s="http://ftb.ca.gov/partnerservice/"
      ID="myRequestBody">
      <!--Parameters passed with the method call-->
    </s:GetMyAccountBalances>
  </SOAP-ENV:Body>
```

Here are the simple points about the above listing that will help you understand WSS syntax:

1. The `SOAP:Envelope` element contains namespace declarations for SOAP, WSS, and XMLDS.

2. The `SOAP:Header` element contains just one child element (`wsse:Security`), which is the wrapper for all the security information. The `wsse:Security` element has two child elements, namely a `wsse:BinarySecurityToken` element and a `ds:Signature` element.
3. The `wsse:BinarySecurityToken` element contains a security token. A security token is like a security pass or an identity card that you are required to show if you want to enter a restricted access area. There are several types of electronic security tokens.

The most popular and widely used security token is a login-password pair, like the one you use while checking your e-mail.

A login-password pair is a human readable security token. There are some security tokens that are in binary form (and therefore not necessarily human readable). Such tokens are referred to as binary security tokens. For example an X509 certificate (a widely popular format for digital certificates developed by ITU-T) is a binary security token.

The `ValueType` attribute of the `wsse:BinarySecurityToken` element tells what type of binary security token is wrapped inside this `BinarySecurityToken` element. The `ValueType` attribute contains `wsse:X509v3` as its value, which identifies X509 certificates.

The `EncodingType` attribute of the `wsse:BinarySecurityToken` element tells the encoding of the binary security token. As already explained, it is not possible to wrap binary data inside XML format as such. Therefore, we have to encode binary data (usually as a sequence of base-64 encoded values) before wrapping inside XML. The X509 certificate is wrapped inside the `wsse:BinarySecurityToken` element as the element content.

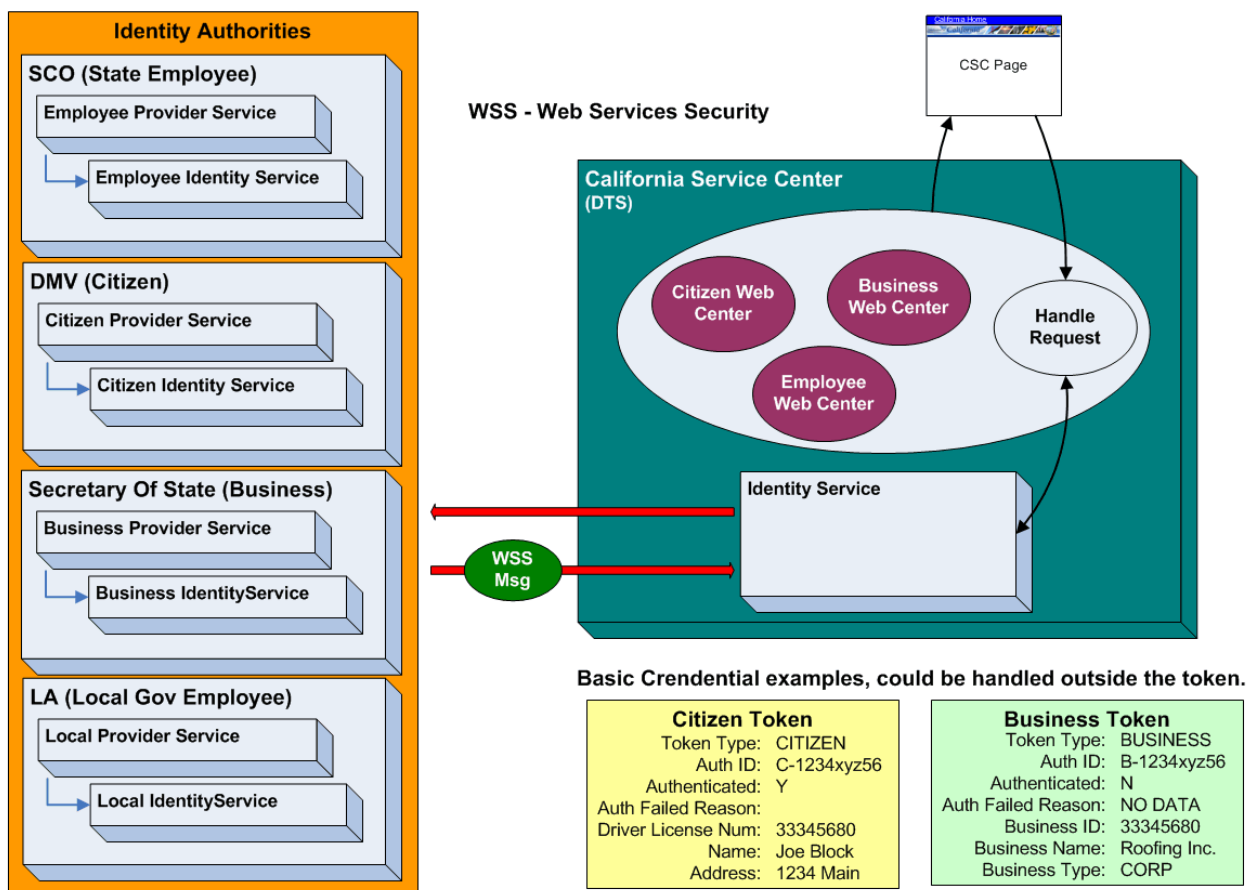
4. The `ds:Signature` element is the same as the one we discussed in the section on XML signatures. Note two important things:
 - Look at the `URI` attribute of the `Reference` element. Its value (`#myRequestBody`) is a fragment identifier that points toward the `SOAP:Body` element. This means that the `SOAP:Body` element is the one that we have signed and wrapped the signature in XMLDS tags.
 - Secondly, also look at what the `ds:KeyInfo` element contains. It is a `wsse:SecurityTokenReference` element. The `wsse:SecurityTokenReference` element contains references to security tokens. In our case, it has a child element named `wsse:Reference`, whose `URI` attribute points toward the `wsse:BinarySecurityToken` element discussed in point 3 above. This means that the public key inside the X509 certificate (which the `wsse:BinarySecurityToken` element wraps) will be used to verify the signature.

Identity and Authentication

Authentication means verifying the identity of a user. When you check your e-mail, you enter your username and password to get authenticated. It is assumed that you have kept your password confidential. Therefore the knowledge of your password is used to make sure that you are the one who is trying to check your email.

Similarly, one can use certificates as authentication tokens. Recall from the previous example that an X509 certificate was wrapped within the SOAP header of the `GetMyAccountBalances` method call. The certificate was actually a security token (just like a password) that the recipient of the WSS message can use in order to authenticate the user before allowing specially discounted rates for booking.

To simplify identity management, a single department could be designated to handle identities of the same type. For example, State Controllers Office might handle State employees, DMV citizens, Secretary of State business users, and Los Angeles County local government. A single identity service could run at a centralized location (California Service Center in the example below) which would determine if identity needed to be established based on the selected user interaction then invoke the appropriate identity authority.



Identity Authorities Pattern

Sharing of Authentication Information

A security token is presented to a gatekeeper in order for a user to get authenticated. Now imagine that the gatekeeper is guarding the main gate of a large building with many offices. Visitors are required to show their ID cards and get authenticated at the main gate. The gatekeeper checks the ID card by matching it with his internal record and then allows the visitor to enter the building.

Let's suppose that you want to visit several offices in the building. Each office has an entrance with a gatekeeper guarding the entrance of each office. You need to get authenticated at the entrance of each office. The gatekeeper at the entrance of each office repeats the same authentication act.

What if individual offices in the building trust the authentication performed by the gatekeeper of the main gate? The building will become a trusted domain, of which each individual office will be a part. Naturally if this type of trust exists between different offices, they would like to share the processing load of the authentication act.

A possible solution to allow sharing of authentication information is to issue a temporary identification badge to a visitor at the main gate of the building. The gatekeeper at the main gate will issue a badge to each visitor after successful authentication. The identification badge will have a short expiry. The visitor will show the identification badge while entering each office. The office gatekeeper will check the validity of badge before allowing or disallowing a person to enter the office.

Such scenarios are common in Enterprise Application Integration. Whether applications are running within or across the boundaries of an enterprise, the sharing of authentication information forms an important part of application integration effort. Naturally, the sharing of authentication information prevents each application from having to perform the entire authentication process.

Security Access Markup Language

SAML is an XML vocabulary that defines the syntax necessary to exchange identity information between applications. The identity information is exchanged in the form of assertions. A security provider service is responsible for providing assertions about its trusted partners and therefore acts as an *SAML* assertion authority.

For example, the California Service Center (CSC) might request an assertion from the Secretary of State Business Provider Service (a *SAML* authority). CSC is a *requester application* and the *subject* of the assertion as well. After getting the assertion from the provider service, CSC will wrap the assertion in a WSS message and send the WSS message to the appropriate department application. The receiving department will rely on the assertion to decide whether to allow access to its application. The receiving department is a *relying party*.

Notice that the *SAML* specification does not define any security attributes by itself. *SAML* users are expected to design their own security attribute namespaces.

In the following listing, we have wrapped an *SAML* assertion in a WSS message.

In order to understand what information this listing contains, you need to compare it with the previous listing in [#An Introduction to Web Service Security](#). There are some differences between the two:

1. There is no `BinarySecurityToken` element in the following listing. Instead of a security token, we have an assertion. The `Assertion` element appears as a child of the `wsse:Security` element, just like the `BinarySecurityToken` element in the first listing.
2. There are two `ds:Signature` elements in the following listing. The first appears within the `Assertion` element. The *SAML* authority produced this signature while issuing the assertion, so that any application who receives this assertion can verify its integrity. We have not shown the details of this signature for the sake of simplicity.

The second `ds:Signature` appears as a direct child of the `wsse:Security` element. This signature is from our *SAML* authority, which produced the signature over the `GetMyAccountBalances` element in the SOAP body while authoring the request.

Compare the `ds:Signature` element in previous listing with the `ds:Signature` element in the following listing. Both these `Signature` elements were produced by the application. The one difference is their `ds:KeyInfo` elements.

In the previous listing, the `ds:KeyInfo` element referred to the certificate wrapped inside the `BinarySecurityToken` element. But in this the following listing, there is no `BinarySecurityToken` element. Instead, we have an `Assertion` element acting as a security token. Therefore it makes sense to refer to the assertion from the `ds:KeyInfo` element.

3. As already explained, the `ds:KeyInfo` element in the following listing refers to the assertion. When the message reaches the relying party, they will need to validate the signature in order to verify requester's identity as well as the integrity of the message. Therefore the recipient will need a public key to verify the signature. Where is the public key that the application can use to verify the signature? The `Assertion` element is the most relevant place to look for the public key.

There is only one key inside the `Assertion` element. Its name is "MyKey". The application will use this key to verify the signature.

4. Notice the `SubjectConfirmation` element within the `Assertion` element, which specifies the relationship between the subject and the author of the message that contains the assertion.

The `SubjectConfirmation` element should specify that the subject authored the message that contains this assertion. The `SubjectConfirmation` element has two child elements, namely a `SubjectConfirmation` and a `ds:KeyInfo` element. The two child elements form a pair.

The `ConfirmationMethod` element wraps the string identifier for the holder-of-key method that we discussed earlier. The holder-of-key method simply specifies that the author of this message is the subject of the assertion and it holds the key wrapped by the accompanying `ds:KeyInfo` element. Notice that the accompanying `ds:KeyInfo` element, which is a sibling of the `ConfirmationMethod` element, wraps the key named "MyKey"

I have already said that the tour operator uses the same key (named `MyKey`) to sign the `GetMyAccountBalances` element. This provides a link between the WSS message author and the subject of the assertion. The application will simply need to verify the integrity of the assertion (by verifying the signature of the SAML authority) and the signature of the requesting application. If the two signatures validate, the recipient application can be sure that the assertion is not fake and it is really asserting the author of the WSS message.

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP:Envelope
  xmlns:SOAP="http://www.w3.org/2001/12/soap-envelope"
  xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <SOAP:Header>
    <wsse:Security>
      <saml:Assertion
        xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
        MajorVersion="1"
        MinorVersion="0"
        AssertionID="http://ftb.ca.gov/AuthenticationService/SAMLAassertions/786"
        Issuer="http://ftb.ca.gov">
```

```

IssueInstant="2003-03-11T02:00:00.173Z">
<Conditions
  NotBefore="2003-03-11T02:00:00.173Z"
  NotOnOrAfter="2003-03-12T02:00:00.173Z"/>
<AttributeStatement>
  <Subject>
    <NameIdentifier
      NameQualifier="http://ftb.ca.gov"
      MyTourOperator
    </NameIdentifier>
    <SubjectConfirmation>
      <ConfirmationMethod>
        urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
      </ConfirmationMethod>
      <ds:KeyInfo>
        <ds:KeyName>MyKey</ds:KeyName>
        <ds:KeyValue> ... </ds:KeyValue>
      </ds:KeyInfo>
    </SubjectConfirmation>
  </Subject>
  <Attribute
    AttributeName="CitizenStatus"
    AttributeNamespace="http://ftb.ca.gov/AttributeService">
    <AttributeValue>TaxLevel5</AttributeValue>
  </Attribute>
</AttributeStatement>
<ds:Signature>...</ds:Signature>
</saml:Assertion>
<ds:Signature>
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <ds:Reference URI="#myRequestBody">
      <ds:Transforms>
        <ds:Transform
          Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        </ds:Transforms>
        <ds:DigestMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <ds:DigestValue>BSDFHJYK21f...</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>
      GKLLKAJFLASKJ52kjKJKLJ345KKKJ...
    </ds:SignatureValue>
    <ds:KeyInfo>
      <wsse:SecurityTokenReference
        <wsse:KeyIdentifier wsu:id="SAML786Identifier"
          ValueType="saml:Assertion">
          http://ftb.ca.gov/AuthenticationService/SAMLAassertions/786
        </wsse:KeyIdentifier>
      </wsse:SecurityTokenReference>
    </ds:KeyInfo>
  </ds:Signature>
</wsse:Security>

```

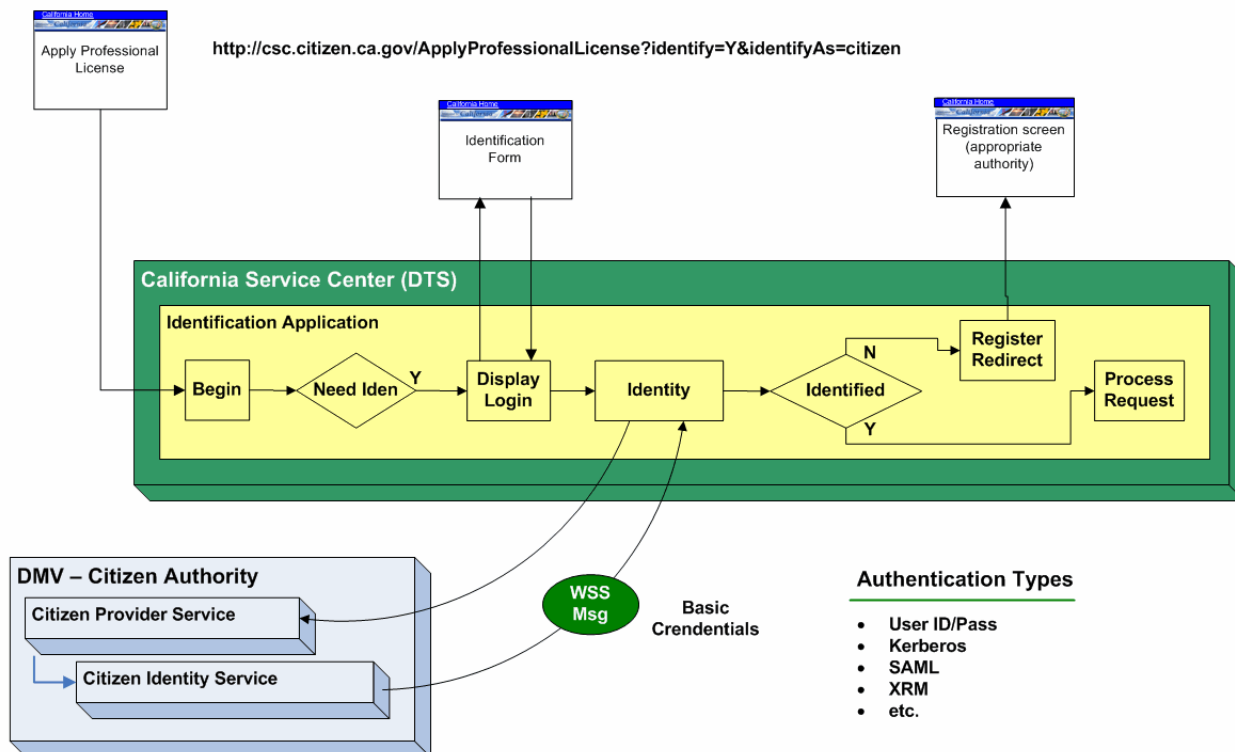
```

</SOAP:Header>
<SOAP-ENV:Body>
  <s: MyAccountBalances
    xmlns:s="http://ftb.ca.gov/partnerservice/"
    ID="myDiscountRequestBody">
    <!--Parameters passed with the method call-->
  </s: MyAccountBalances>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

A Citizen Request Example

Let's illustrate the above security principles in an example scenario. In this case, a citizen will apply for a professional license (doctor, dentist, real estate, CPA, etc.). They will first go to the new California Service Center which will have a link (or picture, or button) they can click on to start the online application process.

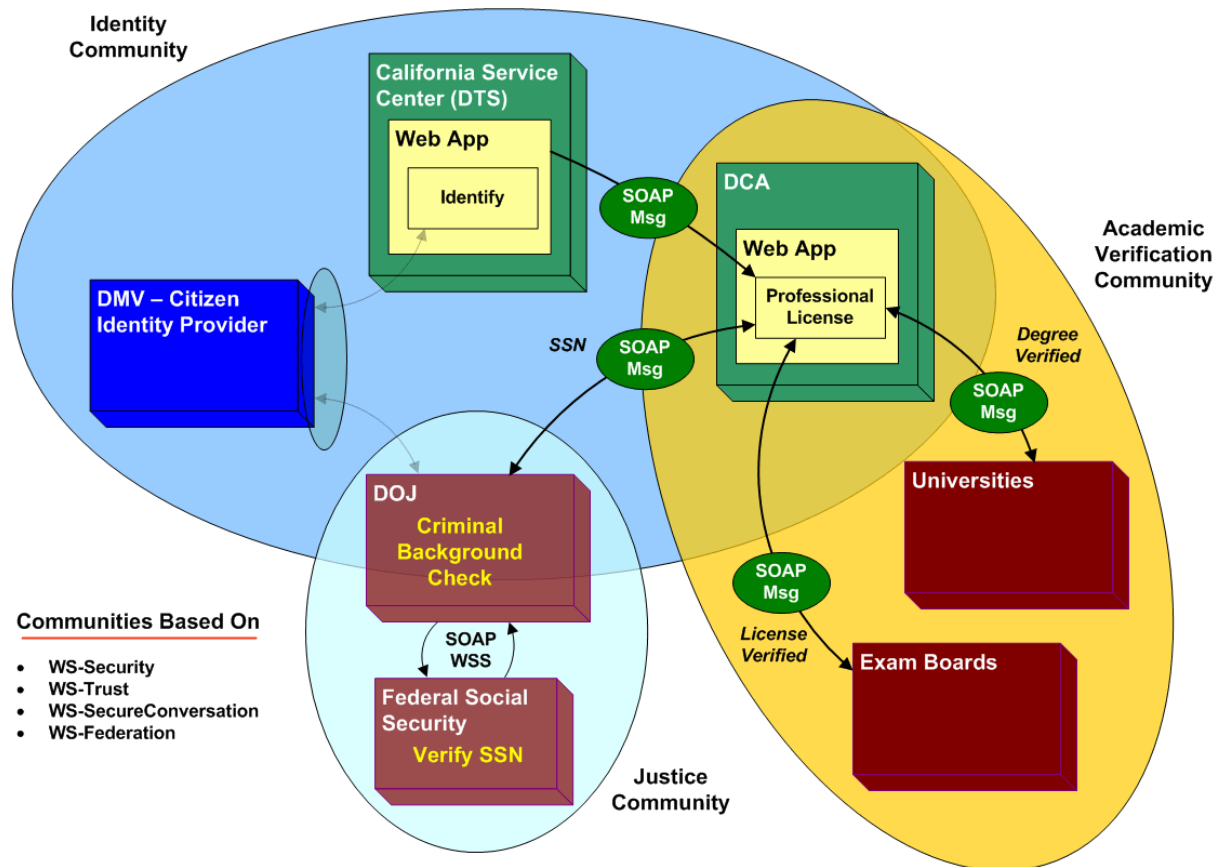


Notice, the onclick action is encoded with two parameters: `identify=y` and `identifyAs=citizen`. The CSC web application will check these parameters and invoke the Identity process if required. In this case, because we want to be identified as a Citizen, a request will be sent to the DMV Citizen Provider Service – which, for the purpose of this illustration, we have designated DMV as the Authority for identifying citizens within the State.

In reality, one would not embed the above parameters in the URL since they would appear as clear text on the browser status line. Rather, they would be hidden form variables and sent via the HTML POST method.

The Citizen Provider Service will return a WSS Msg (Web Services Security message) containing the basic credentials for this person. It will be up to the appropriate architects with the State to define the exact details of basic credentials. The CSC web application will then redirect the request to the appropriate department application to handle the actual professional license application process.

Let's say CSC invoked a Professional License web application managed by DCA. The DCA application would look at the SOAP message and then send a request to the DMV Citizen Provider Service. It would receive a return message containing the authentication status and basic credentials.



Communities of Interest Pattern

The DCA Professional License application would then consume services at a University to verify that this person had the appropriate degree. It would also consume a service at the appropriate Exam Board to verify that this person had a proper license. Additionally, it would consume the Criminal Background Check service at DOJ, which might send a message to the Federal Social Security Administration Verify SSN service to confirm the SSN.

Notice there are three circles of trusts. In the above example because DMV, DTS, DCA and DOJ are part of the Identity Circle of Trust, DCA and DOJ will use the identity information initiated by DTS and provided by DMV. Further, because Universities and Exam Boards and in the Academic Verification Circle of Trust, they will use the identity information passed to them by DCA.

However, if DOJ were to invoke a service at an Exam Board that required identification, it would not be honored because they are not part of the same circle of trust. So, re-identification would be required.

Also keep in mind that different types and levels of security can be used. That is, one might use certificates, PKI, Kerberos, or other types of tokens. They are all part of the WSS standards.

SOA Firewalls for Web Security

SOAP and XML expose a new attack surface to your organization that could potentially let intruders penetrate to the core of your crucial business services. Packet-level firewalls can't help you secure Web services traffic because they can't detect SOAP and XML traffic. For example, because SOAP typically uses HTTP or SMTP, it easily passes through traditional firewalls—a phenomenon known as the port 80 problem. So, just when you thought firewalls had matured a new kind of “firewall” has appeared: the Edge Enforcement Agent.

Like HTML, XML is a markup language that provides a platform-independent standard for exchanging information between systems on the intranet and Internet. XML differs from HTML, however. HTML is static: It provides a finite set of ways to structure text information. When new needs arise, the HTML standard must be updated to accommodate them. In contrast, XML is a more abstract markup language that provides built-in extensibility through a schema that you define.

XML provides a way to format or structure data and commands or transaction requests. Two applications that support the same XML schema can easily exchange data and request transactions. But although XML lets you assemble a message, it doesn't address getting the message from the client to the server and back again. That task is the job of a protocol—SOAP, in the case of Web services.

SOAP gives applications a way to send XML-based messages over a network within HTTP or SMTP. When one application needs another application's services, the first application formats a service request (i.e., a function name and parameters) into XML, then packages the request in a SOAP envelope and sends it. The target application opens the envelope, executes the request, then uses SOAP to return a response. Environments such as Windows .NET Framework let the application developer work at a high level of abstraction, but the Framework still relies heavily on SOAP and XML, so related security concerns still come into play.

Because of XML's platform-independent nature and its ability to let disparate systems interface easily, most Web services use well-known XML schemas and consequently are vulnerable to a much broader variety of potential attacks than are narrower technologies such as Distributed COM (DCOM) and EDI. As a result, you face a greater likelihood of people sniffing the data, non-authenticated clients directly connecting to and trying to retrieve data from your Web services server, and Denial of Service (DoS) attacks that use malformed messages to exploit a well-known schema.

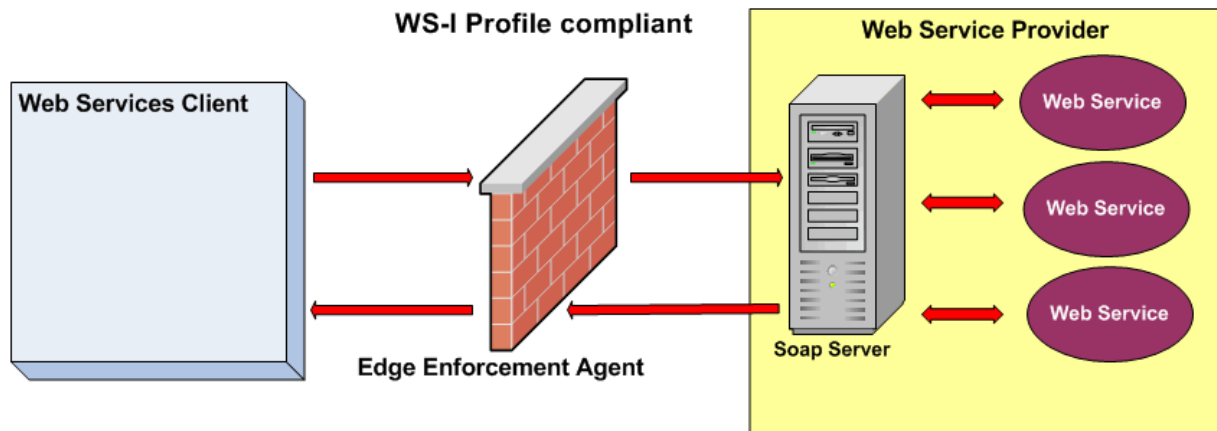
Traditional firewalls, which look at the world in terms of IP addresses, ports, and protocols, address risks that occur at a much lower level than the level at which SOAP and XML reside. Instead of determining whether to pass a given packet to the internal network, SOA firewalls validate traffic in terms of Web services, individual messages, and data elements and evaluate whether to let a given requester access a specific operation. XML-embedded malware, such as worms, Trojan horses, and DoS attacks, are risks with SOAP and XML.

You can address SOAP/XML security concerns three ways. First, if your use of SOAP/XML is light and limited to a stable set of partners, you might be able to get by with a classic firewall. However, the vendor must enhance the firewall so that it can at least recognize SOAP within HTTP and other protocols. You can then enable SOAP and XML content between your organization and its trusted business partners and block everything else.

A second option for SOAP/XML firewalls is to build your own. Although probably not an appealing alternative for most organizations, building your own firewall is possible, and tools exist to help you do the job. For example, Microsoft Internet Security and Acceleration (ISA) Server 2000 lets you write Internet

Server API (ISAPI) filters on an ISA server, and Microsoft provides a model ISAPI filter for validating SOAP/XML messages while they're at the ISA server.

The third, and usually best, option is an application-level SOA firewall that operates behind your classic firewall to validate only SOAP/XML traffic. Similar to a proxy, this type of product receives the Web service message as though the application-level firewall were actually the Web service. These products inspect the message; authenticate the person, program, or organization that sent it; then verify that the sender is authorized to the Web service and the requested operation. Authentication can use a simple username and password, a certificate, or a federated system that uses Security Assertion Markup Language (SAML).



The Edge agent must look inside the SOAP/WSS messages and enforce security access to the SOAP server.

An Edge Enforcement Agent can authenticate credentials against sources such as a Lightweight Directory Access Protocol (LDAP) directory (e.g., Active Directory—AD) or a Remote Authentication Dial-In User Service (RADIUS) server. Then, the agent checks the requested Web service and operation and the data elements (i.e., parameters) within the message to make sure the request is valid and authorized for the user. Either before or after authentication, depending on the product, the agent weeds out malformed messages and DoS attacks by ensuring that the request's format complies with the corresponding schema. The agent forwards messages that pass these checks to the appropriate Web service.

Most agents also provide some type of audit functionality and logging so that you can monitor what's happening with your Web services. Because encryption and XML parsing are CPU-intensive, this more complex proxy architecture is important to implementing SOA firewalls in high-security and high-volume Web service scenarios. Because SOAP/XML supports security at the transport level, a SOA firewall can use Secure Sockets Layer (SSL) and Transport Layer Security (TLS) to encrypt the entire HTTP-based message stream.

But sometimes you need to be able to encrypt or digitally sign portions of an XML document—to facilitate multiparty transactions, for example. The XML Encryption and XML Signature security standards meet these intra-document cryptography needs. Because a SOA firewall functions as a proxy Web service, all authentication, encryption, and decryption take place at the firewall, letting you centrally and consistently control authentication, encryption, and policy checks even if Web services are scattered on servers throughout your network. Another advantage is that, because only decrypted traffic can be inspected, encrypted content is decrypted at the firewall and compared against the firewall's policy.

California SOA Center of Excellence

Introduction

A successful state-wide SOA program will require both centralized and federated components. Singular vision & goals, governance, enterprise repository management, and several operational functions (certification lab, UDDI repository, maintain service reference model, service help desk, and search taxonomy) should be centralized. Service development (and possibly some SOA operations) should be federated to the producing departments. In some of the above functions, centralized does not mean single instance. For example one would establish at least two UDDI repositories for scalability and accessibility.

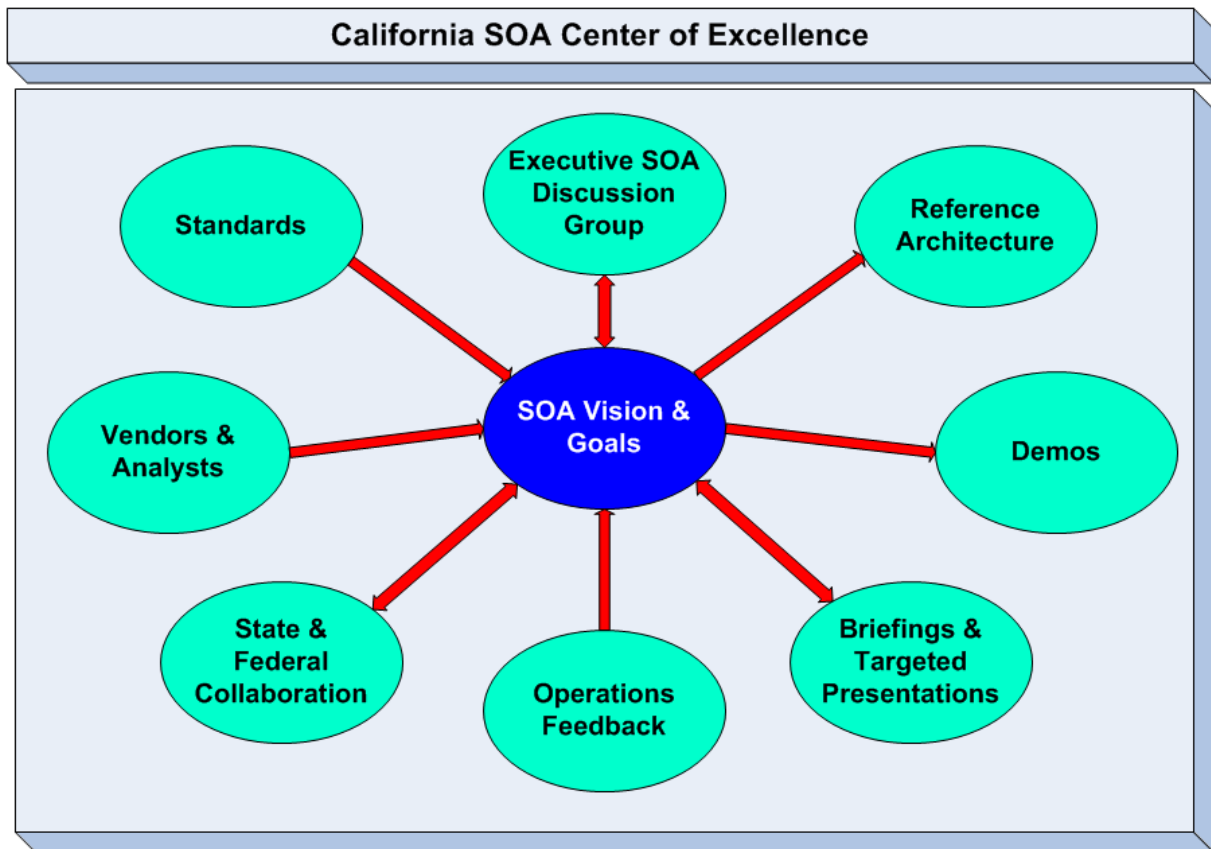
Because SOA components are designed for enterprise use, there are a number of critical governance and operational issues that need to be addressed. Such as:

- How will developers be supported?
- How will business architects and technical architects determine which components already exist?
- How will enterprise components be mapped to business services?
- How will component versioning and release packaging be controlled?
- How will components be certified?
- How will components be tested for performance, availability, and scalability?
- How will component usage be tracked?
- How will developers locate code for an existing service?
- How will enterprise components be promoted and marketed?
- How will service contract compliance among data centers be handled?
- Will there be a centralized help desk?
- How will enterprise troubleshooting be handled?
(A composite service might consist of 4 services, each running in a different data center.)
- Will there be a “reference architecture” based on industry best practices?
- Will there be demo applications?
- Will there be a state-wide search service utilizing a common language? (a taxonomy)

It is obvious that there is a very important need for an oversight group to act as the center hub for the enterprise. It is recommended that a California SOA Center of Excellence be established to fulfill many of these tasks. This group would lead the SOA effort, be the “go to” experts for departmental business service implementers, facilitate discussion groups, lead collaboration efforts, build a reference architecture based on best practices, and host demos. They could also maintain a performance lab to ensure that critical enterprise components will meet performance and availability expectations, manage a centralized help desk, handle compliance escalations via an expert distributed architecture technical staff, and define a state-wide search taxonomy.

SOA leadership cannot be static. Rather, it should evolve as standards mature and change, vendors products change based on market dynamics and changing standards, analysts revise their best practices, and government politics and regulations change.

So the first set of responsibilities of the California SOA Center of Excellence is shown in the following SOA Excellence diagram.



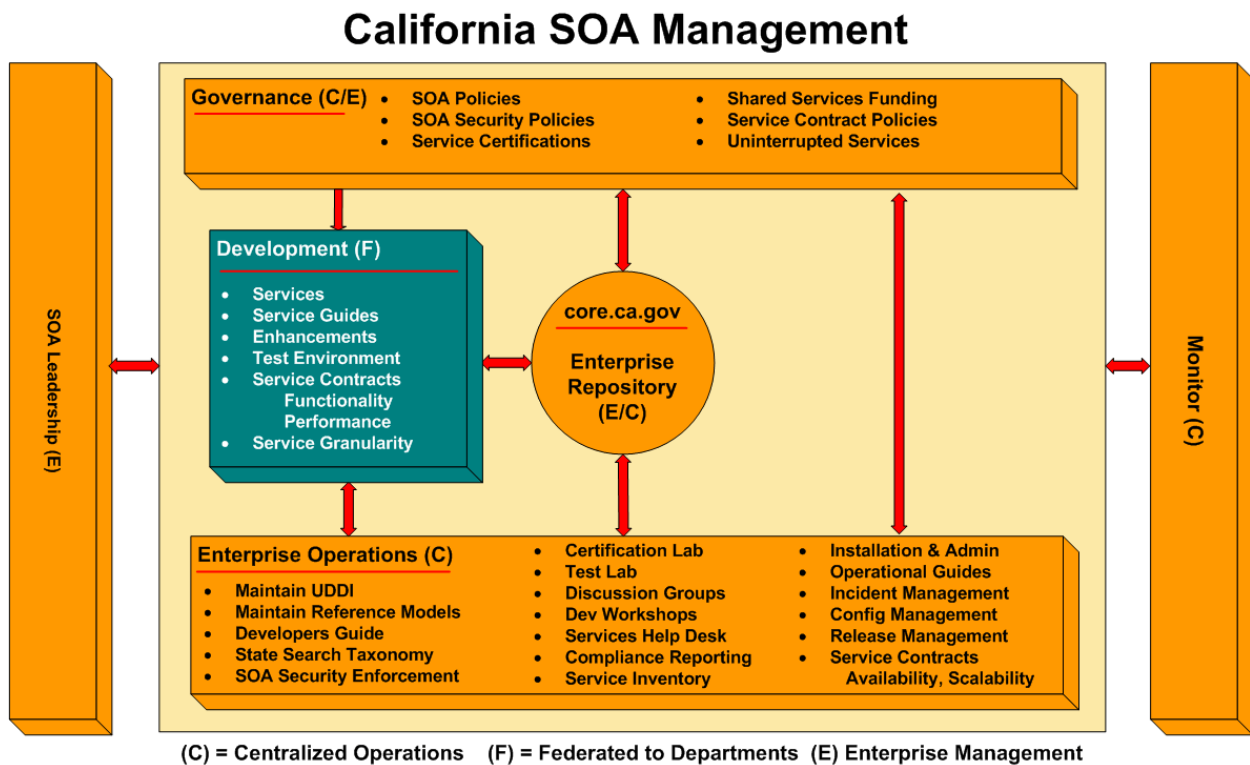
SOA Excellence Model

1. *Standards* – SOA is comprised of many standards which are managed by several standard bodies. It will be very important to stay on top of standard developments since they will have a large impact on SOA architecture. Attending conferences, monitoring discussion groups, and regularly checking W2C, OASIS, and WS-I websites are all key activities.
2. *Vendors & Analysts* – Relationships with key vendors is important in order to keep abreast of product developments and vendor visions for SOA. Subscribing to industry analysts newsletters is also a good way of staying informed.
3. *State & Federal Collaboration* – The Federal Enterprise Architecture Group and its ancillary operations set the standard for states to follow. Additionally, many states are in the process of implementing SOA-based models. So, ongoing collaboration with other states makes good sense.
4. *Operations Feedback* – The SOA vision and goals must be practical. So, feedback from SOA operations must be factored into the vision and goals must be updated accordingly.
5. *Briefings & Targeted Presentations* – SOA means different things to different people. So, preparing presentations that are targeted to a particular group will be most effective. For example, executives, business architects, technical architects, and IT developers all have different priorities within SOA. So, presenting SOA in terms of specific audience type would be most effective.

5. *Demos* – An SOA Center of Excellence is a great place to demonstrate the SOA environment. Demos can be built to support the presentations targeted at specific audiences. Often, it is easier to understand a point by witnessing a demo. Plus, a demo proves out an environment and allows “what if” scenarios.
6. *Reference Architecture* – An SOA environment should exist as the model for developers. That is, a reference implementation that incorporates key SOA components would provide developers platform-specific components to measure their services against. The Reference Architecture would include both J2EE and .NET components.
7. *Executive SOA Discussion Group* – Sponsoring an SOA discussion group for executives would be one way of providing executives with an easy mechanism for exchanging thoughts and ideas.

SOA Management Model

Service components will be built and tested by individual departments (for example DMV, FTB, BOE, DTS, etc). Each service will be submitted for certification. Upon approval, the service must go through the data center deployment process. The recommended model for SOA Management is Federated Development with Centralized Operations.



This does not imply a single data center. Rather, it means a small number of data centers will interoperate and possibly back each other up with regard to shared services. This implies that shared services might be deployed in a “hot standby” mode to ensure uninterrupted services across data centers.

SOA Centralized Functions

1. Governance

- *Service Contract Policies* - Service performance contracts will be published by a producing department for a given web service. Consuming organizations will build their applications around these contracts. The contract process itself will be established by the Governance portion of SOA Management. Actual service performance will be monitored by the individual data centers. They will either work the problem directly or collaborate with the producing development organization to fix the problem. Additionally, they will notify governance personnel of services that are out of compliance. Consuming organizations will have a key part in revising performance contracts.
- *Service Certification* - Certifying web services will be a key function. All new and revised services must go through the certification process to ensure that they will play nicely in the enterprise architecture. The testing will be done in the Certification Lab maintained by Enterprise Operations.
- *SOA Policies and SOA Security* - Based on SOA Leadership input, Governance and Enterprise Operations will establish and enforce SOA Policies and SOA Security Policies.
- *Shared Services Funding* – Effective SOA Management will require an investment in staff, hardware, software, and tools. Therefore, an appropriate budget must be granted to support Governance, Enterprise Operations, and maintenance of the Enterprise Repository (core.ca.gov). Some outside staff (consultants) will likely be required to manage certification and performance labs, as well as serve as enterprise technical troubleshooters.
- *Uninterrupted Services* – Policies must be put into place to ensure availability of shared services. This will require an effective failover strategy which probably includes deploying shared services in multiple data centers in “hot standby” mode. These should be on separate networks and in different physical locations.

2. Enterprise Repository (core.ca.gov)

- Reference models, portfolio and application information will be managed in a centrally maintained California Enterprise Repository (core.ca.gov). This includes ensuring that the repository is always available and it is regularly backed up. A guide on how to search and use the repository will also be published.
- The repository will have a hierarchical structure for the BRM, SRM, TRM, and DRM. That is, service components will be related to their business services and the technical components that support them. The owner, description, version and interface definition of each service will be clearly stated as well as any dependencies. Extensive search capabilities will be provided to ensure easy access by developers as well as business architects.

3. Enterprise Operations

- *UDDI Registry* - A state-wide UDDI registry will be maintained by Enterprise Operations for the purpose of locating web services.
- *Maintain Repository Models* – Enterprise Operations would have overall responsibility for core.ca.gov.
- *Developer's Guide* - A developer's guide will be created that states the general guidelines for developing web, shared, and enterprise services.

- *State Search Taxonomy* – A state-wide common search language (taxonomy) should be designed in collaboration with all interested parties. The California Service Center Enterprise Search Service searches would utilize this taxonomy.
- *Certification Lab* – Development organizations will package their unit tested services and submit them to Governance for certification. A certification lab will be set up and published to all development organizations so they know how their services will be tested. It will take significant collaboration to initially set up this lab as well as ongoing fine-tuning. The goal should be to ensure that services play nicely together in a distributed environment, meet their stated requirements, have stable and well defined interfaces, and meet all stated security requirements.
- *Test Lab* – This lab would be used to test performance, probe for security vulnerabilities, as well as verify failover processes.
- *Discussion Groups* – Enterprise Operations will facilitate a discussion group for service developers and operational administrators. Note, discussion
- *Developer Workshops* – Enterprise Operations will design appropriate workshops for developers. At minimum, there should be a workshop for developing a base service, and one for developing a composite service. Workshops should implement best practices as defined by SOA Leadership and Governance.
- *Service Help Desk* – A single help desk would be responsible for handling all shared service customer issues. A comprehensive help database could be built over time to make this job easier. A single owner for a given problem would increase customer satisfaction.
- *Compliance Reporting* – Some shared services will have contracts specifying availability, scalability, and recoverability metrics. Enterprise Operations will be responsible for reporting out of compliance services.
- *Installation and Administration* – SOA data centers will be responsible for installing, configuring, deploying, and registering their services. They will also ensure that proper logging is turned on and review the operational logs on a regular basis. Appropriate data and files must be backed up on a regular schedule that fits the particular service. Additionally, the services must be installed and configured to meet performance, availability, and scalability requirements.
- *Service Inventory* – SOA data centers will be responsible for updating the Enterprise Repository with information about their services. At minimum, this will include the service owner, version number, and who is using the service. This last data element might be dynamically updated by service monitoring tools.
- *Incident Management* – When a service problem arises, Enterprise Operations is expected to resolve the problem in an efficient and timely manner. There will be a single owner for each ticket which should increase customer satisfaction. This probably falls under the ITSM umbrella.
- *Configuration Management* – When enhancement or bug fixes are applied by a development organization, the resulting service will be versioned and resubmitted to the certification group.

Upon successfully certification, the newly version component will be put into production via Enterprise Operation's configuration management policy (again, probably ITSM based).

- *Release Management* – Initial services, as well as major changes to existing services, will be provided in a release package and submitted for certification. Upon approval, the release package will be deployed into production by Enterprise Operations following proper release policies.
- *SOA Security Enforcement* – Some services will not have security requirements while others may have very stringent requirements. Services that participate in Identity, Access, and Privacy implementations must follow specific enterprise security policies. This is especially true for those services that are part of a “circle of trust”. While the Governance group will determine the policies, they will be enforced by Enterprise Operations. Note, fraudulent penetration software should be deployed by CISO personnel and used to test data center shared services on a regular basis.
- *Service Contracts* – Enterprise Operations will be responsibility for ensuring availability, scalability, and recoverability requirements are met as defined in a services contract. They evaluate data gathered by monitoring tools to determine whether they are in or out of compliance. They will provide compliance reports to as well as be responsible for getting a service back into compliance. They may use an escalation process if they need additional help.
- *Operational Guides* – Enterprise Architecture will provide operational guides detailing startup, shutdown, and service recovery procedures. They will contain configuration and deployment packaging information. A section on common error messages and typical troubleshooting procedures would also be helpful.
- *Gather Operational Data* – A common set of tools will be specified by Governance and Enterprise Operations. It is expected that Enterprise Operations will be proactive in evaluating the data generated by the tools and take appropriate action when potential problems are indicated.

SOA Federated Functions

1. Service Development

- *Services* – The prime function of development is to produce web services. It is also their responsibility to ensure that enterprise requirements are taken into consideration not just department requirements. Additionally, they must be fully tested, documented, and packaged for easy and successful deployment. Development personnel must work with the certifying organization to ensure proper certification prior to releasing to the SOA data center.
- *Service Guides* – Developers will provide a guide for consumers of their service. At minimum, the public interface will be detailed as well as examples of how to use the service.
- *Enhancements* – Developers must collaborate on enhancement requests. Upon agreement, a schedule would be provided for incorporating the new functionality by the designated development organization.
- *Service Contracts* – Developers will be responsibility for meeting functionality and performance requirements as specified in a service contract.

- *Test Environment* – Each development organization will maintain a suitable test environment to prove their service meets all enterprise requirements regarding functionality and performance.
- *Services Granularity* – This is a key component in determining how manageable the SOA environment will be, as well as the degree of service reuse. If the service interfaces are too complex or if there are simply too many services, then manageability will become a real problem. Services need to be easily composed into higher level services to achieve maximum reuse. So, careful thought and ongoing diligence will be required.

Appendix A - Federal SOA

http://www.cio.gov/documents/CIOC_AIC_Service_Component_Based_Architectures_2.0_FINAL.pdf

The Federal Architecture and Infrastructure Committee along with the Federal CIOs Council produced the Federal Enterprise Architecture which is based on SOA and Web Services.

"An architecture that provides for reuse of existing business services and rapid deployment of new business capabilities based on existing capital assets is often referred to as a service-oriented architecture (SOA). " -- Federal CIOs Council

Service Components

"At the top of the service component hierarchy is a federation of business components. Federal business components contain multiple business processes or services that can be shared across agencies. At the lower levels are lower granularity service components that implement elements of the processes or services. Components at the lower levels are selected and integrated to build higher-level services. Relating the service component hierarchy to the current FEA reference models provides the linkage from the efforts at the lines of business level in the BRM and the supporting services in the SRM. "

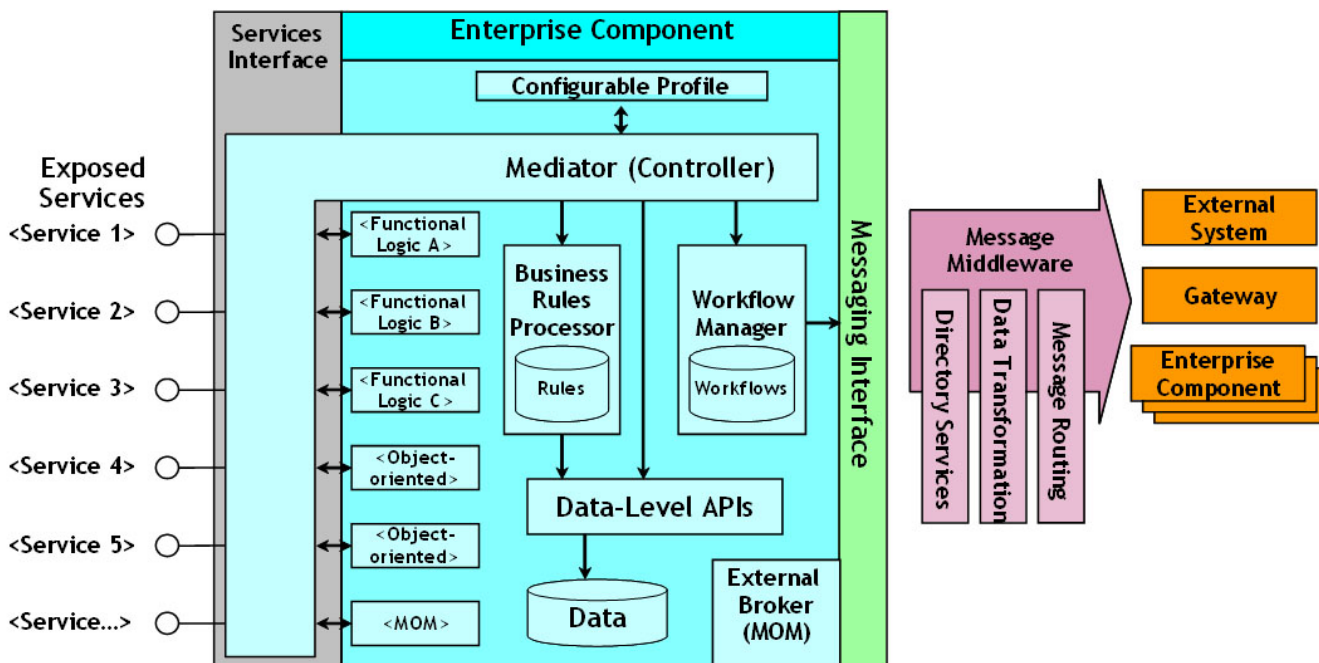


Figure 2-3 - A Notational Federal Enterprise Component

Component Architecture Description Standards

"An effective service component-based architecture requires the adoption of and adherence to technical standards that help promote a common understanding of the use case of service components in an effective manner. A key goal in describing service component architectures is to provide a clear separation of concerns between the functional and the operational aspects of the

architecture. A unified modeling language-based notation, as illustrated in figure 2-4, is one way to describe a service component. "

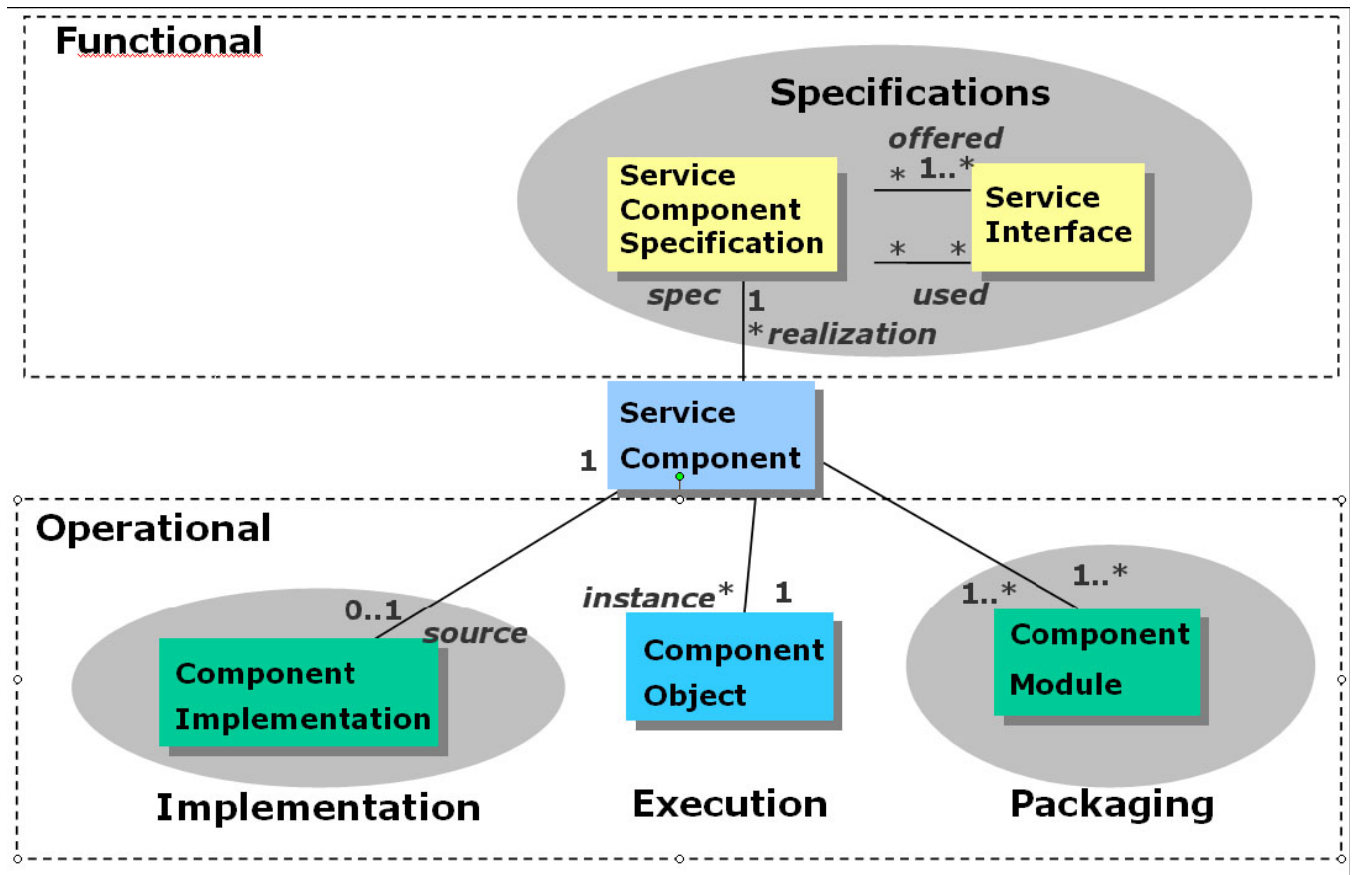


Figure 2-4 - A Notational UML Service Component Description

Federal Service Component Harvesting and Provisioning Model

"The FEA reference models are designed to be used to identify and define reusable service components and service component interfaces, and the definition of the federal service component granularity roughly corresponds to the level of granularity of an IT 300 Exhibit. Reference models are particularly used for business analysis and capital planning. Figure 2-5 illustrates the notional relationships of reference models to typical SOA activities. "

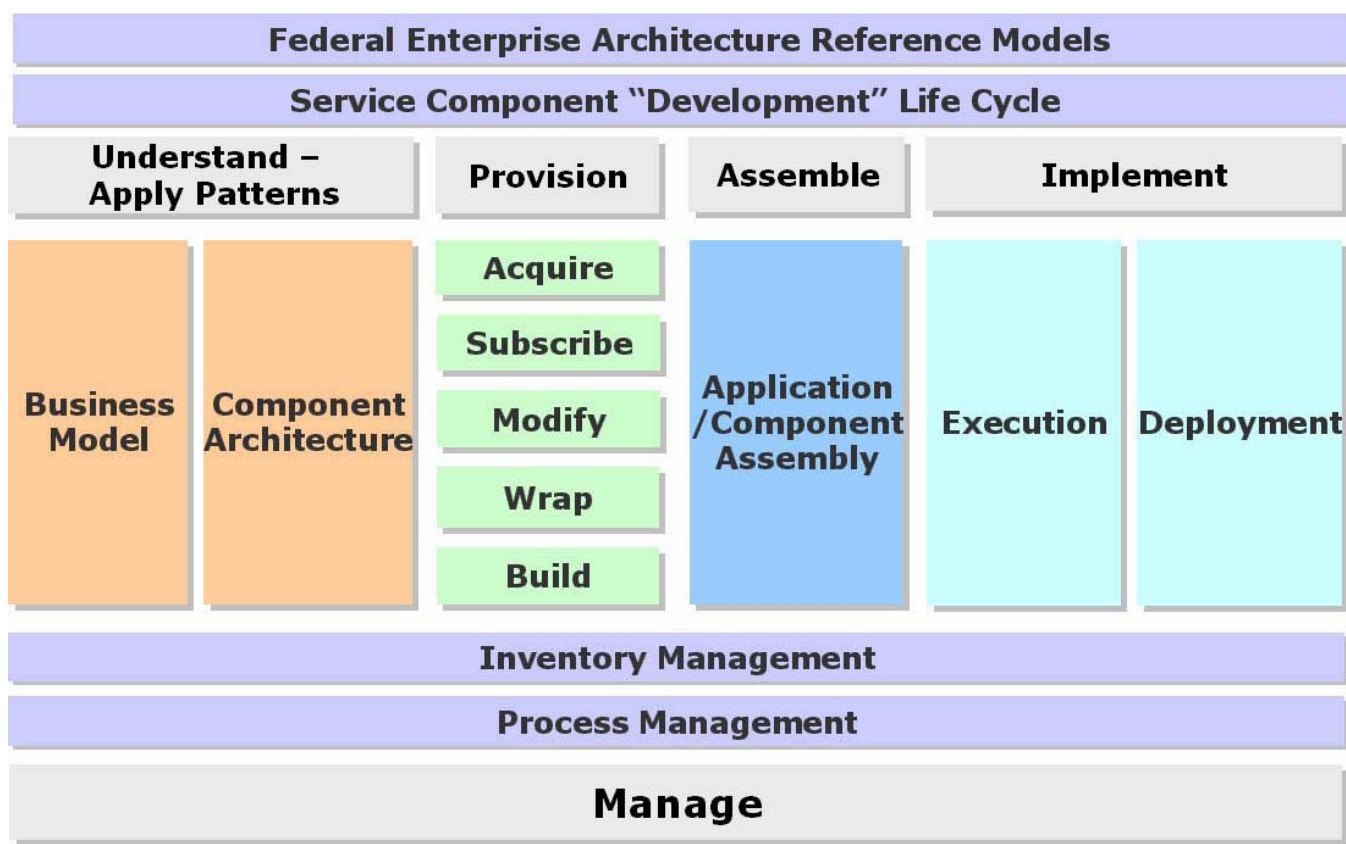


Figure 2-5 - Notational SOA and Interaction with FEA Reference Models

Business Modeling

"SOA activities start with business modeling, with the BRM and SRM acting as the source for structuring capabilities. The BRM and SRM are designed to bridge the gap between capital planners and enterprise architects, allowing expression of the business services and concepts that are required to support the operation of the organization. "

Service Component Architecture

"The content gathered during the business modeling phase is communicated via work products or artifacts to formulate the solution outline. At this macro-level design phase, the solution may be communicated via high-level service component models to develop a better understanding of the business domain and the solution. Service components are defined and the dependencies between collaborating service components are clearly identified. The business services exposed by these service component interfaces support the operational needs of the organization. Non-functional requirements are also taken into account to determine the constraints that must be applied to the solution. The business services are typically defined in a technology-neutral reference model, like the BRM or SRM. "

Component Provisioning and Assembly

"Service component and asset consumers and producers follow separate processes geared to their respective needs of creating or harvesting business solutions and high-quality service components. Solution developers (asset consumers) examine existing service components and as-

sets to potentially harvest them as reusable assets. Component producers (provisioners), on the other hand, look at requirements to be the basis for creating components or acquiring service components for reuse. "

Federal Enterprise Component Registry

"After potential federal enterprise components have been identified, it is important to have a way for enterprise architects to manage the service component interface profile, publish the profile, and provide the methodology for accessing the component and other key component information. In addition, it is important to be able to understand the strategic, tactical, and operational attributes of the service component. "

"Equally important are the relationships between service components that are being used, because a failure in one component used by many federal agencies could result in a cascading catastrophic system failure of business processes that depend on the functionality of that service component. "

"Because of the complexities associated with service component-based architectures and SOA, it is very important that mission- or business-critical federal components be properly certified. In order to have a scaleable certification process for federal components, it is important to establish the criteria for service component certification. It follows that the registry/repository concept for Federal components will also be based on a hierarchical certification governance process. "

"The FEA reference models are designed to be used to identify and define reusable service components. Service components, by design, separate the services they provide from the way those functions are implemented. This is true at all levels of the service component hierarchy. Service component-based architectures, if properly implemented, provide a framework to achieve a very high ROI for federal IT assets. The FEA SRM serves as the foundation for federal enterprise architects and capital planners to better serve the citizen by utilizing a service component-based architectural approach to federal IT asset management. "

Federal Enterprise Architecture Management System <http://www.feapmo.gov>

"FEA analysis and maintenance are greatly facilitated through the use of an Internet-based automated EA repository and analysis tool—the Federal Enterprise Architecture Management System (FEAMS). Agencies will be given access to FEAMS and can use it in both capital planning and architecture development efforts. "

"In addition to storing the FEA reference models, FEAMS will include general information on agencies' IT initiatives. Initiative alignment to the BRM Lines of Business that they support, the service components and technology that these components leverage, and the performance metrics that they use in achieving performance objectives will be presented. "

Service Component Registry/Repository and Collaboration www.core.gov

"The creation of a component repository and registry for service components is envisioned to be one of the tangible, ongoing outcomes of the FEA analysis at the service, technology, and data layers. As reusable service components are identified and harvested, and collaboration between agencies begins to take place, there will be the need for a collaboration-based repository for storing, maintaining, and sharing these service components. "

Appendix B - Web Service Tenets (Microsoft)

The following four tenets are frequently cited:

Tenet 1: Boundaries Are Explicit

Services interact through explicit message-passing over well-defined boundaries. Crossing service boundaries may be costly, depending upon geographic, trust, or execution factors. A boundary represents the border between a service's public interface and its internal, private implementation. A service's boundary is published by means of WSDL and may include assertions dictating the expectations of a given service. Crossing boundaries is assumed to be an expensive task for several reasons, some of which are listed below:

1. The physical location of the targeted service may be an unknown factor.
2. Security and trust models are likely to change with each boundary crossing.
3. Marshalling and casting of data between a service's public and private representations may require reliance upon additional resources—some of which may be external to the service itself.
4. While services are built to last, service configurations are built to change. This fact implies that a reliable service may suddenly experience performance degradations due to network reconfigurations or migration to another physical location.
5. Service consumers are generally unaware of how private, internal processes have been implemented. The consumer of a given service has limited control over the performance of the service being consumed.

The [Service-Oriented Integration](#) pattern tells us that "service invocations are subject to network latency, network failure, and distributed system failures, but a local implementation is not. A significant amount of error detection and correction logic must be written to anticipate the impacts of using remote object interfaces." While we should assume that crossing boundaries is an expensive process, we must also exercise caution in the deployment of local methods designed to minimize such boundary crossings. A system that implements monolithic local methods and objects may gain performance but duplicate functionality of a previously defined service (this technique was referred to as "cut and paste" in OOP and shares the same risks regarding versioning of the service).

There are several principles to keep in mind regarding the first Tenet of SO:

- Know your boundaries. Services provide a contract to define the public interfaces it provides. All interaction with the service occurs through the public interface. The interface consists of public processes and public data representations. The public process is the entry point into the service while the public data representation represents the messages used by the process. If we use WSDL to represent a simple contract, the `<message>` represents the public data while the `<portType>` represents the public process(es). The article "[Data on the Outside vs. Data on the Inside](#)" examines these issues in greater detail.
- Services should be easy to consume. When designing a service, developers should make it easy for other developers to consume it. The service's interface (contract) should also be designed to enable evolving the service without breaking contracts with existing consumers. (This topic will be addressed in greater detail in future papers in this series.)

- Avoid RPC interfaces. Explicit message passing should be favored over an RPC-like model. This approach insulates the consumer from the internals of the service implementation, freeing service developers to evolve their services while minimizing the impact on service consumers (encapsulation by using public messages instead of publicly available methods).
- Keep service surface area small. The more public interfaces that a service exposes, the more difficult it becomes to consume and maintain it. Provide few well-defined public interfaces to your service. These interfaces should be relatively simple, designed to accept a well-defined input message and respond with an equally well-defined output message. Once these interfaces have been designed they should remain static. These interfaces provide the "constant" design requirement that services must support, serving as the public face to the service's private, internal implementation.
- Internal (private) implementation details should not be leaked outside of a service boundary. Leaking implementation details into the service boundary will most likely result in a tighter coupling between the service and the service's consumers. Service consumers should not be privy to the internals of a service's implementation because it constrains options for versioning or upgrading the service. The Anti-Patterns section of this paper provides a detailed example of this issue.

Tenet 2: Services Are Autonomous

Services are entities that are independently deployed, versioned, and managed. Developers should avoid making assumptions regarding the space between service boundaries since this space is much more likely to change than the boundaries themselves. For example, service boundaries should be static to minimize the impact of versioning to the consumer. While boundaries of a service are fairly stable, the service's deployment options regarding policy, physical location, or network topology are likely to change.

Services are dynamically addressable through URIs, enabling their underlying locations and deployment topologies to change or evolve over time with little impact upon the service itself (this is also true of a service's communication channels). While these changes may have little impact upon the service, they can have a devastating impact upon applications consuming the service. What if a service you were using today moved to a network in New Zealand tomorrow? The change in response time may have unplanned or unexpected impacts upon the service's consumers. Service designers should adopt a pessimistic view of how their services will be consumed—services will fail and their associated behaviors (service levels) are subject to change. Appropriate levels of exception handling and compensation logic must be associated with any service invocation. Additionally, service consumers may need to modify their policies to declare minimum response times from services to be consumed. For example, consumers of a service may require varying levels of service regarding security, performance, transactions, and many other factors. A configurable policy enables a single service to support multiple SLAs regarding service invocation (additional policies may focus on versioning, localization, and other issues). Communicating performance expectations at the service level preserves autonomy, since services need not be familiar with the internal implementations of one another.

Service consumers are not the only ones who should adopt pessimistic views of performance—service providers should be just as pessimistic when anticipating how their services are to be consumed. Service consumers should be expected to fail, sometimes without notifying the service itself. Service providers also cannot trust consumers to "do the right thing." For example, consumers may attempt to communicate using malformed/malicious messages or attempt to violate other policies necessary for successful service interaction. Service internals must attempt to compensate for such inappropriate usage, regardless of user intent.

While services are designed to be autonomous, no service is an island. A SOA-based solution is fractal, consisting of a number of services configured for a specific solution. Thinking autonomously, one soon realizes there is no presiding authority within a service-oriented environment—the concept of an orchestration "conductor" is a faulty one (further implying that the concept of "roll-backs" across services is faulty—but this is a topic best left for another paper). The keys to realizing autonomous services are isolation and decoupling. Services are designed and deployed independently of one another and may only communicate using contract-driven messages and policies.

As with other service design principles, we can learn from our past experiences with OO design. Peter Herzum's and Oliver Sims's work on Business Component Factories provides some interesting insights on the nature of autonomous components. While most of their work is best suited for large-grained, component-based solutions, the basic design principles are still applicable for service design.

Given these considerations, here are some simple design principles to help ensure compliance with the second principle of SO:

- Services should be deployed and versioned independently of the system in which they are deployed and consumed.
- Contracts should be designed with the assumption that once published, they cannot be modified. This approach forces developers to build flexibility into their schema designs.
- Isolate services from failure by adopting a pessimistic outlook. From a consumer perspective, plan for unreliable levels of service availability and performance. From a provider perspective, expect misuse of your service (deliberate or otherwise), and expect your service consumers to fail—perhaps without notifying your service.

Tenet 3: Services Share Schema and Contract, Not Class

As stated earlier, service interaction should be based solely upon a service's policies, schema, and contract-based behaviors. A service's contract is generally defined using WSDL, while contracts for aggregations of services can be defined using BPEL (which, in turn, uses WSDL for each service aggregated).

Most developers define classes to represent the various entities within a given problem space (for example, Customer, Order, and Product). Classes combine behavior and data (messages) into a single programming-language or platform-specific construct. Services break this model apart to maximize flexibility and interoperability. Services communicating using XML schema-based messages are agnostic to both programming languages and platforms, ensuring broader levels of interoperability. Schema defines the structure and content of the messages, while the service's contract defines the behavior of the service itself.

In summary, a service's contract consists of the following elements:

- Message interchange formats defined using XML Schema.
- Message Exchange Patterns (MEPs) defined using WSDL.
- Capabilities and requirements defined using WS-Policy.
- BPEL may be used as a business-process level contract for aggregating multiple services.

Service consumers will rely upon a service's contract to invoke and interact with a service. Given this reliance, a service's contract must remain stable over time. Contracts should be designed as explicitly as possible while taking advantage of the extensible nature of XML schema (xsd:any) and the SOAP processing model (optional headers).

The biggest challenge of the Third Tenet is its permanence. Once a service contract has been published it becomes extremely difficult to modify it while minimizing the impact upon existing service consumers. The line between internal and external data representations is critical to the successful deployment and reuse of a given service. Public data (data passed between services) should be based upon organizational or vertical standards, ensuring broad acceptance across disparate services. Private data (data within a service) is encapsulated within a service. In some ways services are like smaller representations of an organization conducting e-business transactions. Just as an organization must map an external Purchase Order to its internal PO format, a service must also map a contractually agreed-upon data representation into its internal format. Once again our experiences with OO data encapsulation can be reused to illustrate a similar concept—a service's internal data representation can only be manipulated through the service's contract. Pat Helland examines several issues related to public and private data representations in "[Data on the Outside vs. Data on the Inside](#)."

Given these considerations, here are some simple design principles to help ensure compliance with the third principle of SO:

- Ensure a service's contract remains stable to minimize impact upon service consumers. The contract in this sense refers to the public data representation (data), message exchange pattern (WSDL), and configurable capabilities and service levels (policy).
- Contracts should be designed to be as explicit as possible to minimize misinterpretation. Additionally, contracts should be designed to accommodate future versioning of the service through the extensibility of both the XML syntax and the SOAP processing model.
- Avoid blurring the line between public and private data representations. A service's internal data format should be hidden from consumers while its public data schema should be immutable (preferably based upon an organizational, defacto, or industry standard).
- Version services when changes to the service's contract are unavoidable. This approach minimizes breakage of existing consumer implementations.

Tenet 4: Service Compatibility Is Based Upon Policy

While this is often considered the least understood design tenet, it is perhaps one of the most powerful in terms of implementing flexible Web services. It is not possible to communicate some requirements for service interaction in WSDL alone. Policy expressions can be used to separate structural compatibility (what is communicated) from semantic compatibility (how or to whom a message is communicated).

Operational requirements for service providers can be manifested in the form of machine-readable policy expressions. Policy expressions provide a configurable set of interoperable semantics governing the behavior and expectations of a given service. The [WS-Policy](#) specification defines a machine-readable policy framework capable of expressing service-level policies, enabling them to be discovered or enforced at execution time. For example, a government security service may require a policy enforcing a specific service level (Passport photos meeting established criteria must be cross-checked against a terrorist identification system, for example). The policy information associated with this service could be used with a number of other scenarios or services related to conducting a background check. WS-Policy can be used to

enforce these requirements without requiring a single line of additional code. This scenario illustrates how a policy framework provides additional information about a service's requirements while also providing a declarative programming model for service definition and execution.

A policy assertion identifies a behavior that is a requirement (or capability) of a policy subject. (In the scenario above the assertion is the background check against the terrorist identification system.) Assertions provide domain-specific semantics and will eventually be defined within separate, domain-specific specifications for a variety of vertical industries (establishing the WS-Policy "framework" concept).

While policy-driven services are still evolving, developers should ensure their policy assertions are as explicit as possible regarding service expectations and service semantic compatibilities.

Appendix C - SOA Best Practices (IBM)

1. Establish a champion or executive sponsor for SOA.
2. Divide the enterprise into business components (cohesive activities which collaborate with other business components).
3. Develop a SOA strategy which defines the business context, pain points, reference architecture and a living roadmap for SOA adoption for a line of business and/or enterprise.
4. Assign service domain owners and implement governance mechanisms to ensure that a corporate SOA strategy gets implemented in delivered and acquired applications.
5. Extend systems development methodology to address creation of business services with corresponding design attributes for services.
6. Encapsulate key existing/legacy functionality as services, as appropriate.
7. Favor large grained services that align with business process boundaries.
8. Compose atomic services into coarse-grained business services.
9. Build for consumability; refactor services so that they are as broadly applicable as practical.
10. Use top down and bottom up analysis to create business services which removes redundancy and creates opportunities for services.

Appendix D - SOA Advantages (Patricia Seybold Group)

Brenda M. Michelson, Sr. VP

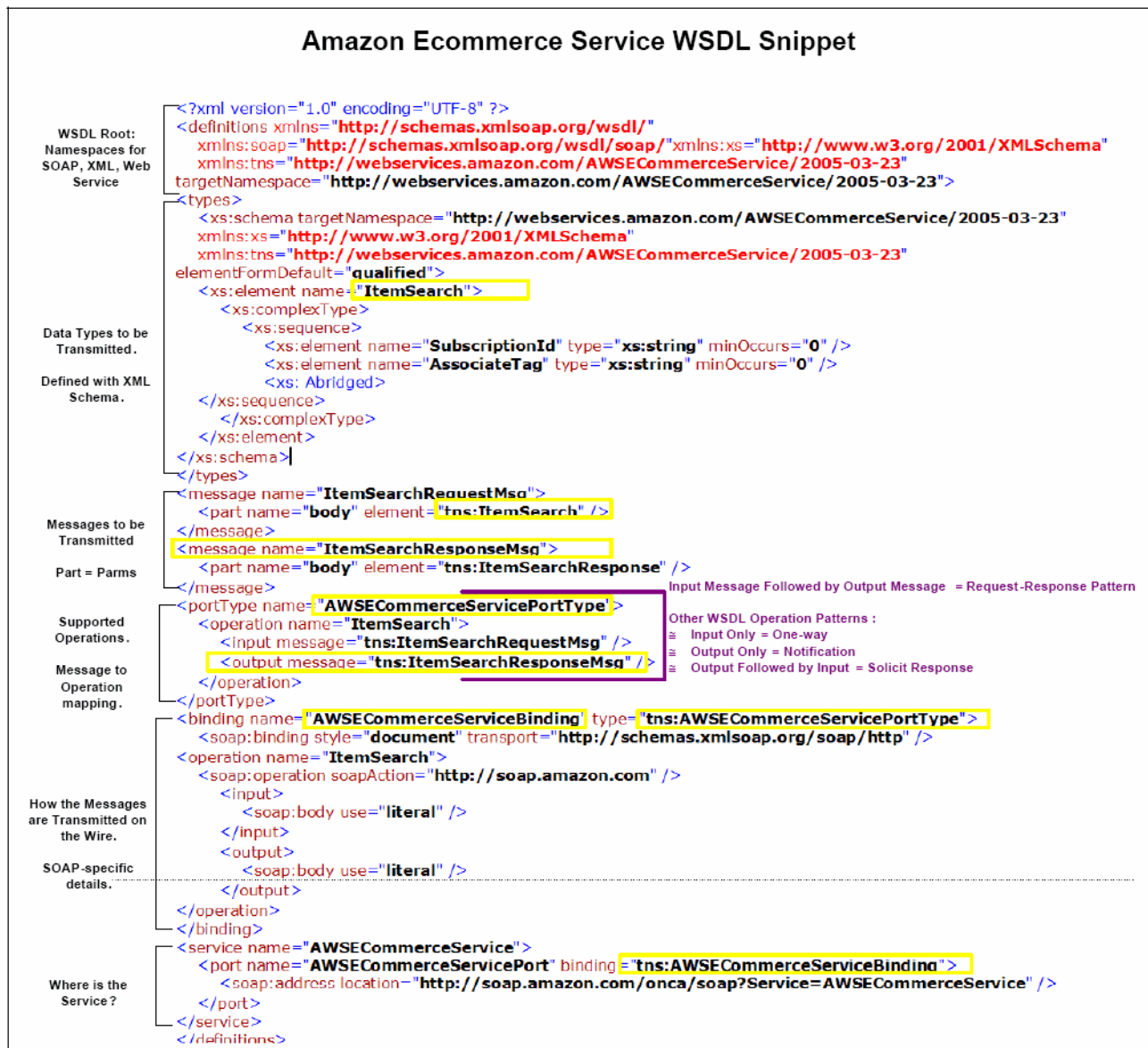
SOA Business Advantages:

- *Consistent Experience.* An SOA can provide a consistent experience for customers and partners across channels and lines of business.
- *Business Agility.* An SOA can add new functionality, expose functionality to new channels, and vary functionality based on context (customer, partner, entry point).
- *Mix and Match.* An SOA can compose business solutions from a reusable service collection, leveraging internal and external services.
- *Optimize Interactions.* An SOA can optimize business interactions for customers, partners, and internal constituents through the implementation of business scenarios (process, events, and services) versus traditional applications.

SOA IT advantages:

- *Reduction of Costs.* Reuse of services reduces IT development and maintenance time and costs.
- *Leverage Existing IT Investments.* Your service providers are existing code (objects, components, legacy modules, and application package APIs) and information assets (databases, files, and documents).
- *Transition Strategies.* An SOA can provide application and portfolio transition strategies.

Appendix E – WSDL Example



This is an abridged version of the WSDL for Amazon's E-commerce Service (ECS). The annotations on the left hand-side describe the WSDL sections. The yellow boxes within the WSDL highlight key elements, such as data elements, messages and bindings, and their recurrence in the various sections.

- Patricia Seybold Group

Appendix F – Legacy Integration Patterns

Overview

Since mainframe applications will be around for a long time and many departments depend on the mainframes for their mission critical applications, the state must plan to integrate mainframe applications into the new SOA-based environment. Departments should take a close look at their application portfolios and devise an application maturity plan which would should separate them into categories such as don't modify, wrap with web service interfaces, reengineer into Java or .NET applications, or plan to retire. Following are brief discussion of three popular patterns.

Integrating Existing Mainframe Apps - Unmodified

There are several ESB products that have a broad range of application integration capability. IBM Websphere ESB, Oracle Fusion, Cape Clear ESB, plus a number of other companies have products in this area. Enterprise Service Bus products not only provide messaging infrastructure for web services, they also provide a variety of adapters to integrate native language interfaces (such as COBOL, CICS, MQ Series, Java, FTP, etc.).

IBM

<http://www-306.ibm.com/software/info1/websphere/index.jsp?tab=landings/esb>

Oracle

<http://www.oracle.com/products/middleware/index.html>

Cape Clear

<http://www.capeclear.com/products/cc6.shtml>

Placing Web Service Interfaces on Existing Mainframe Apps

Makes mainframe applications (particularly Natural and Adabas) look like web services. EntireX executes on the mainframe and exposes the service interfaces. ApplinX solution requires no changes to mainframe code.

SoftwareAG EntireX

<http://www.softwareag.com/Corporate/products/entirex/default.asp>

SoftwareAG ApplinX

<http://www.softwareag.com/Corporate/products/applinx/default.asp>

Compiling COBOL Code into Web Service Languages

Fujitsu Consulting provides a COBOL compiler for a variety of platforms and languages. For example, NetCOBOL for .NET is a COBOL compiler created specifically for Microsoft's .Net Framework. This means that COBOL is just another .NET scripting language (like VB.NET, C#, J#, etc.). This allows COBOL code to be mixed with C# or VB.NET code. It compiles to Microsoft MSIL (language neutral, .NET runtime) code.

NetCOBOL main page

<http://www.netcobol.com/products/>

NetCOBOL for .NET

<http://www.netcobol.com/products/windows/netcobol.html>

Appendix G - Definitions

AJAX: Asynchronous JavaScripting and XML is a web development technique for creating interactive Web pages. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire Web page does not have to be reloaded each time the user makes a change. AJAX is available for Java, .NET, PHP, and other languages. <http://en.wikipedia.org/wiki/AJAX>

Architecture: Representation of the structure of a system that describes the constituents of the system and how they interact with each other.

Application Architecture: Representation of an application and its parts, their inter-relationships and functions.

AVDL: Application Vulnerability Description Language. <http://www.oasis-open.org/specs/index.php#avdlv1.0>

BPEL: Business Process Execution Language for Web Services provides a means to formally specify business processes and interaction protocols. BPEL provides a language for the formal specification of business processes and business interaction protocols. By doing so, it extends the Web Services interaction model and enables it to support business transactions. BPEL defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces. <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>

Business Component: Represents the implementation of an autonomous business concept, business service, or business process. It consists of all the technology elements (i.e., software, hardware, data) necessary to express, implement, and deploy a given business concept as an autonomous, reusable element of a large information system. It is a unifying concept across the development lifecycle and the distribution tiers.

Business (Domain) Component: Organizational unit that offers business services operation based on rules of that business.

Business Component System: Set of cooperating business components assembled together to deliver a solution to a business problem.

Business Logic Component: Software unit that offers small-grained business logic that has a large degree of reuse throughout the organization. Sub-components that manage and execute the set of complex business rules that represent the core business activity supported by the component.

CAP: Common Alerting Protocol. <http://www.oasis-open.org/specs/index.php#capv1.0>

Component: Independently deployable unit of software that exposes its functionality through a set of services accessed via well-defined interfaces. A component is based on a component standard, is described by a specification, and has an implementation. Components can be assembled to create applications or larger-grained components.

Component Architecture: Internal structure of a component described in terms of partitioning and relationships between individual internal units.

Component-Based Architecture: Architecture process that enables the design of enterprise solutions using large service components. The focus of the architecture may be a specific project or the entire enterprise. This architecture provides a plan of what needs to be built and an over-view of what has been built already.

Component Registry: Application designed to provide a directory of available components based on profile and or specification. Registries usually provide efficient mechanisms for searching for components in multiple ways, such as by service, price, and/or provider.

Component Repository: Application designed to store component specifications and implementations. Often provides facilities to efficiently search for and retrieve components for evaluation against desired component specifications though the search capabilities may be off-loaded to a component registry.

CORBA: Common Object Request Broker Architecture, created by the Object Management Group (<http://www.omg.org/>) vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.

COTS Components: Commercial Off the Shelf (COTS) components that can satisfy business process and data requirements for large functional domains or lines-of-business. Examples of COTS components would be Enterprise Resource Planning (ERP) products such as those as offered by commercial software companies.

Data: Factual or numerical business information of record that is maintained by the service component. The encapsulated service component is fully responsible for maintaining this information.

Data-Level Application Programming Interfaces: Services internal to the service component that support access to the data of record maintained within the service component. These services may span numerous distributed data sources.

DCOM: Distributed Common Object Model is an extension of the [Component Object Model \(COM\)](#) that allows COM components to [communicate](#) across [network](#) boundaries. DCOM uses the [RPC](#) mechanism to [transparently](#) send and receive information between COM components. Microsoft first introduced it in 1995.

DSML: Directory Services Markup Language. <http://www.oasis-open.org/specs/index.php#capv1.0>

Distributed Component: Lowest level of component granularity. It is a software element that can be called at run-time with a clear interface and a clear separation between interface and implementation. It is autonomously deployable. A distributed component provides low ROI for capital planning purposes.

E-Business Patterns: Patterns for e-business are a group of proven reusable assets that can be used to increase the speed of developing and deploying net-centric applications, like Web-based applications.

ebXML: Electronic Business using eXtensible Markup Language. <http://www.ebxml.org/>

Encapsulation: Hiding implementation details within a component so that an implementation is not dependent on those details.

Enterprise Architecture: Meta-architecture of an organization or the sum of all architectures within an organization.

Enterprise Component: Large-granularity business component of an organization.

Enterprise Service Bus (ESB): A class of integration software that is intended to support the deployment of Web services. An ESB combines messaging, basic transformation, and content-based routing. The inputs and outputs of an ESB are Service Data Objects (SDO).

Extensibility: Ability to extend the capability of a component so that it handles additional needs of a particular implementation.

Federated Business Component: Set of cooperating system-level components federated to resolve the business need of multiple end users often belonging to different organizations.

California Enterprise Component: Very coarse-grained business component of California Government.

Federation: is a collection of realms/domains that have established trust. The level of trust may vary, but typically includes authentication and may include authorization.

Fit-Gap Analysis: Examination of components within the context of requirements and to make a determination as to the suitability of the service component.

Component Granularity: The size of the unit of component under consideration in some context. The term generally refers to the level of detail at which component is considered, e.g. "You can specify the granularity for this service component".

Identity Mapping: is a method of creating relationships between identity properties. Some Identity Providers may make use of id mapping.

Identity Provider: is an entity that acts as a peer entity authentication service to end users and data origin authentication service to service providers (this is typically an extension of a security token service).

ID-FF: Liberty Identity Federation Framework. ID-FF contains the core specifications that allow for the creation of a standardized, multi-vendor, identity federation network. The FF consists of protocols, schema and profiles. <https://www.projectliberty.org/resources/specifications.php>

ID-SIS: Liberty Identity Services Interface Specifications. ID-SIS uses the [ID-WSF \(new window\)](#) and [ID-FF \(new window\)](#) specifications to provide networked identity services, such as contacts, presence detection, or wallet services that depend on networked identity. The SIS contains two specifications: Personal Profile (ID-SIS-PP): and Employee Profile (ID-SIS-EP):

ID-WSF: Liberty Identity Web Services Framework. ID-WSF provides a basic framework of identity services. Such services could be identity service discovery and invocation. The WSF consists of schema, protocols, and profiles. <http://www.projectliberty.org/resources/specifications.php>

Infrastructure Component: Software unit that provides application functionality not related to business functionality, such as error/message handling, audit trails, or security.

Interface: Mechanism by which a component describes what it does and provides access to its services. This is important because it represents the “contract” between the supplier of services and the consumer of the services.

Intellectual Property: A product of the intellect that has commercial value, including copyrighted property such as literary or artistic works, and ideational property, such as patents, appellations of origin, business methods, and industrial processes.

Interface Profile: the sub-component that provides the ability to customize the component for various uses. The profile can be tailored to suit different deployment architectures well as different sets of business rules across enterprises. The interface profile can specify the business rules and workflow that are to be executed when the component is initialized. The profile can specify the architectural pattern that complements the service component.

Java Server Faces: JavaServer Faces technology is a server-side user interface component framework for Java technology-based web applications. JSF offers a clean separation between behavior and presentation. <http://java.sun.com/j2ee/1.4/docs/tutorial/doc/>

Language Class: Class in an object-oriented programming language to build distributed components. This is NOT considered an SRM component. A language class provides very low ROI for capital planning purposes.

Line of Business: A particular kind of commercial or government enterprise; e.g. "human re-sources" "financial management" "wholesale banking".

Message Authentication: is the process of verifying that the message received is the same as the one sent.

Messaging Interface: Linkage from the service component to various external software modules (component, external systems, gateways, etc.) and other service components.

Notional Component: Set of services packaged into a component, derived from requirements definition. A "desired" component, prior to implementation.

Process Component: Software unit that implements the logic of a process.

Realm or Domain: represents a single unit of security administration or trust.

Reuse: Any use of a preexisting software artifact (component, specification, etc.) in a context different from that in which it was created.

SAML: Security Assertion Markup Language. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

Security Token Service (STS): A security token service is a Web service that issues security tokens (see WS-Security and WS-Trust). That is, it makes assertions based on evidence that it trusts, to whoever trusts it. To communicate trust, a service requires proof, such as a security token or set of security tokens, and issues a security token with its own trust statement (note that for some security token formats this can just be a re-issuance or co-signature). This forms the basis of trust brokering.

Sender Authentication: is corroborated authentication evidence possibly across Web service actors/roles indicating the sender of a Web service message (and its associated data). Note that it is possible that a message may have multiple senders if authenticated intermediaries exist. Also note that it is application-dependent (and out of scope) as to how it is determined who first created the messages as the message originator might be independent of, or hidden behind an authenticated sender.

Service: Discrete unit of functionality that can be requested (provided a set of preconditions is met), performs one or more operations (typically applying business rules and accessing a data-base), and returns a set of results to the requester. Completion of a service always leaves business and data integrity intact.

Service-Component: Modularized service-based applications that package and process together service interfaces with associated business logic into a single cohesive conceptual module. Aim of a service component is to raise the level of abstraction in software services by modularizing synthesized service functionality and by facilitating service reuse, service extension, specialization and service inheritance.

Service-Component Reference Model (SRM): Service component-based framework that can provide—independent of business function—a "leverage-able" foundation for reuse of applications, application capabilities, components, and business services.

Service Data Object (SDO): An Enterprise Service Bus concept where all incoming messages are converted into service data objects.

Service Interface: Set of published services that the component supports. These are aligned with the business services outlined in the service reference model.

Service-Level Agreement: A contract or memorandum of agreement between a service provider and a customer that specifies, usually in measurable terms, what services the service provider will furnish. Information technology departments in major enterprises have adopted the idea of writing a service level agreement so that services for their customers (users in other departments within the enterprise) can be measured, justified, and perhaps compared with those of external (sourcing) service providers.

Service-Oriented Architecture: Architecture that provides for reuse of existing business services and rapid deployment of new business capabilities based on existing capital assets.

Services Interface: A logical boundary that permits software services to be defined independent of the service implementation.

Simple Object Access Protocol (SOAP): A simple XML based protocol to let applications exchange information over HTTP. SOAP is a protocol for accessing a Web Service.

Single Sign On (SSO): is an optimization of the authentication sequence to remove the burden of repeating actions placed on the end user. To facilitate SSO, an element called an Identity Provider can act as a proxy on a user's behalf to provide evidence of authentication events to 3rd parties requesting information about the user. These Identity Providers are trusted 3rd parties and need to be trusted both by the user (to maintain the user's identity information as the loss of this information can result in the compromise of the user's identity) and the Web services which may grant access to valuable resources and information based upon the integrity of the identity information provided by the IP.

SPML: Service Provisioning Markup Language. <http://www.oasis-open.org/specs/index.php#capv1.0>

Solution Assembly: Process of implementing a solution by assembling the necessary components into a complete solution. This process often involves additional “glue” code to integrate the assembled components.

Test Harness: Software that automates the software engineering testing process to test the software as thoroughly as possible before using it on a real application. **Trust Domain:** an administered security space in which the source and target of a request can determine and agree whether particular sets of credentials from a source satisfy the relevant security policies of the target. The target may defer the trust decision to a third party thus including the trusted third party in the Trust Domain.

UBL: Universal Business Language. <http://www.oasis-open.org/specs/index.php#capv1.0>

UDDI: Universal Description Discovery Integration. <http://www.uddi.org/>

Web Service: Functionality provided by a service, which is exposed using the Internet (SOAP, HTTP, WSDL, XML, TCP/IP) as the transport mechanism. Can be internally provided as part of a suite of services or can be offered by external organizations.

Web Service for Remote Portlets: A user-facing Web Service that will provide content, marked for display, to a portal or other aggregating Web application. This moves Web Services out from the back-end model layer. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp

Web Service Flow: The process of combining and orchestrating web services into a unique flow. Individual web methods on multiple web services can be invoked in a precise order that meets the specific application flow requirements.

Workflow Manager: Sub-component that enables one component to access services on other components to complete its own processing. The workflow manager determines which external component services must be executed and manages the order of service execution.

Wrapping: Creation of an interface around legacy functionality (code) that exposes the functionality as services via interfaces that conform to a component specification.

WSDL: An XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

WSDM: Web Services Data Management.

WSIF: The Web Services Invocation Framework (WSIF) is a simple Java API for invoking Web services, no matter how or where the services are provided. WSIF allows stubless or completely dynamic invocation of a Web service, based upon examination of the meta-data about the service at runtime. <http://ws.apache.org/wsif/>

WS-Addressing: describes how to specify identification and addressing information for messages.

WS-Authorization: will describe how to manage authorization data and authorization policies.

WS-BusinessActivity: This specification provides the definition of the business activity coordination type that is to be used with the extensible coordination framework described in the WS-Coordination specification. The specification defines two specific agreement coordination protocols for the business activity coordination type: BusinessAgreementWithParticipantCompletion and BusinessAgreementWithCoordinatorCompletion. Developers can use any or all of these protocols when building applications that require consistent agreement on the outcome of long-running distributed activities. <http://www-128.ibm.com/developerworks/library/specification/ws-tx/>

WS-Federation: describes how to manage and broker the trust relationships in a heterogeneous federated environment, including support for federated identities, sharing of attributes, and management of pseudonyms.

Web Services Inspection Language (WSIL): The WS-Inspection specification "defines how an application can discover an XML Web service description on a Web server, enabling developers to easily browse Web servers for XML Web services. WS-Inspection complements the IBM- and Microsoft-pioneered 'Universal Description, Discovery and Integration (UDDI)' global directory technology by facilitating the discovery of available services on Web sites unlisted in the UDDI registries, and builds on Microsoft's SOAP Discovery technology built into Visual Studio .NET.

WS-MetadataExchange: describes how to exchange metadata such as WS-Policy information and WSDL between services and endpoints.

WS-Policy: represents a set of specifications that describe the capabilities and constraints of the security (and other business) policies on intermediaries and endpoints (e.g. required security tokens, supported encryption algorithms, privacy rules) and how to associate policies with services and endpoints. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/securitywhitepaper.asp>

WS-Privacy: will describe a model for how Web services and requestors state privacy preferences and organizational privacy practice statements. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/securitywhitepaper.asp>

WS-Referral: WS-Referral is a protocol that enables the routing strategies used by SOAP nodes in a message path to be dynamically configured. SOAP itself provides a distributed processing model where SOAP messages can have content destined for specific processing nodes. WS-Routing adds to SOAP the capability of describing the actual message path. WS-Referral provides a mechanism to dynamically configure SOAP nodes in a message path to define how they should handle a SOAP message. It is a configuration protocol that enables SOAP nodes to delegate part or all of their processing responsibility to other SOAP nodes. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsreferspecindex.asp>

WS-Reliability: Web Services Reliability (WS-Reliability) is a SOAP-based protocol for exchanging SOAP messages with guaranteed delivery, no duplicate s, and guaranteed message ordering. WS-Reliability is defined as SOAP header extensions, and is independent of the underlying protocol. <http://developers.sun.com/sw/platform/technologies/ws-reliability.html>

WS-ReliableMessaging: this specification describes a protocol that allows messages to be delivered reliably between distributed applications in the presence of software component, system, or network failures. The protocol is described in this specification in an independent manner, allowing it to be implemented using different network transport technologies. To support interoperable Web services, a SOAP binding is defined within this specification. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsrmspecindex.asp>

WS-Routing: WS-Routing is a simple, stateless, SOAP-based protocol for routing SOAP messages in an asynchronous manner over a variety of transports like TCP, UDP, and HTTP. With WS-Routing, the entire message path for a SOAP message (as well as its return path) can be described directly within the SOAP envelope. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsroutspecindex.asp>

WSRP: Web Services for Remote Portlets. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp

WS-SecureConversation: describes how to manage and authenticate message exchanges between parties, including security context exchanges and establishing and deriving session keys. <http://www-128.ibm.com/developerworks/library/specification/ws-secon/>

WS-Security: describes how to attach signature and encryption headers to SOAP messages. In addition, it describes how to attach security tokens, including binary security tokens such as X.509 certificates and Kerberos tickets, to messages. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-security.asp>

WS-Security SAML Token Profile: <http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf>

WS-Transactions and WS-Coordination: describes how to enable transacted operations as part of Web service message exchanges.

WS-Trust: describes a framework for trust models that enables Web services to securely interoperate by requesting, issuing, and exchanging security tokens. <http://webservices.xml.com/lpt/a/ws/2003/06/24/ws-trust.html>

XACML – Extensible Access Control Markup Language. <http://www.oasis-open.org/specs/index.php#capv1.0>